The MetaKit Library

To display a list of topics by category, click any of the contents entries below. To display an alphabetical list of topics, choose the Index button.

Overview

The MetaKit library is a C++ class library to manage persistent structured data.

The following items are a copy of the information in the file "README.TXT":

1. Basic concepts

The MetaKit Library distinguishes the following aspects of data:

STRUCTURE
: The way in which information is organized. This corresponds to concepts like structs / arrays / entries, and databases / tables / records.

REPRESENTATION
: The format used to represent information on disk, in memory, or on screen.

MANIPULATION
: All operations which access or change the current state of available information.

The distinction between structure and representation is an important one, and is similar to the separation between the "logical" and "physical" data models of relational database theory.

2. Run-time data structure

Although C++ can deal with member fields and pointers to them, they are not objects (i.e. instances of a class). This makes it more difficult to design code which deals with member fields in a general way.

How do you pass a set of member fields and implement sorting on multiple keys in a generic way? Such issues are much easier to deal with in dynamically typed languages than in C++. But don't throw C++ out the window just yet...

Welcome to run-time data structures: instead of defining CLASSES of objects to use for your application data, why not use OBJECTS (called "rows" in MetaKit) of a generic class to represent this data, with OBJECTS (called "properties" in MetaKit) to represent the different member fields as well?

This sounds more complicated than it is in actual use.

Example
Here is an example, which will actually compile when using MetaKit:

```
cStringProp pName ("Name");
cIntProp pAge ("Age");

cRow rData;
pName (rData) = "John Williams";
pAge (rData) = 45;

cView vNames;
vNames.Add(rData);

    // sorting a view with one entry is not very useful...
cView vNames2 = vNames.Sort(pAge);

int age = pAge (vNames2 [0]);
```

As you can see in this code, the operators for "()" and "[]" have been overloaded to introduce a reasonably natural notation for property access and for view indexing, respectively. Note how the generic sort routine is told what to sort on (this is not trivial - **Sort** actually relies on *pAge* to tell it how to compare values).

LISTS OF PROPERTIES

With properties as ordinary C++ objects (and a nasty overload of the comma operator), a whole range of application-independent extensions become feasible. Here's how to sort a view on two properties:

```
vNames = vNames.SortOn((pAge, pName));
```

There is another version of **Sort** which takes a second list of properties as parameter, specifying which properties should be sorted in descending instead of ascending order. To sort by decreasing age:

```
vNames = vNames.SortOnDown((pAge, pName), (pAge));
```

CONSTANT VALUES

Other operator overloads make it possible to define a row with a specified property and value in a single step:

```
cRow rData2 = pName ["Julian Bream"];
```

Rows can be concatenated, so you can write:

```
vNames.Add(pName ["Julian Bream"] + pAge [39]);
```

SELECTION CRITERIA

With constant values, selection becomes a one-liner (this one will produce an empty view, given the current entries in *vNames*):

```
cView vNames3 = vNames.Select(pAge [45] + pName ["Paco Pena"]);
```

Another general function selects on a range of criteria:

```
vNames3 = vNames.SelectRange(pAge [21], pAge [65]);
```

SUMMARY

Run-time data structures make it possible to implement algorithms which can be used in a wide range of applications. With care, and a few devious operator overloads (of ",", "()", and "[]"), this mechanism can be embedded in C++ without sacrificing readability too much.

3. Run-time data representation

Data representation is a complex issue: how do I keep track of an item of information - on disk, in memory, and on the screen? As an example, should my loop index be a short, long, unsigned short, string, or a pointer? Where do I make that decision? What consequences does it have?

Languages such as C++ require you to deal with this decision up front, i.e. when declaring variables, data members, but also argument types and function results. Weakly typed languages (SmallTalk, Lisp, Icon) will check to make sure that an appropriate type is used at run-time, but lose all type checking opportunities during compilation. Some languages simply use text strings for everything (PERL, TCL, HyperTalk).

One way to deal with the problem in C++, is to choose safe limits (long ints, large string buffers, double precision floats). Although this may be fine for many in-memory structures, it is often not an acceptable choice for data stored in a disk file, in a database, or transmitted over a data-communication channel.

Since MetaKit is based on the strongly typed C++ language, it must also deal with this issue.

The approach to this problem in MetaKit, is to use safe limits for the library interface, introducing a limited set of "domains", with automatic conversion to and from different internal representations. The conversion is performed by customizable "data handlers", which implement a wide range of run-time storage representations. In other words, you have to decide whether to use integers when defining a property, but you do not need to decide on its allowed range.

The central message here, is that the application should be concerned only with the domain of data (integer, string, floating point, ...), not with the best way to store it. Even if a data value looks like text, acts like text, and quacks like text, that does not mean that it has to be STORED as text (it might be a numeric resource id, to use an example). The representation might even be changed over time.

The distinction between domain and representation is not easy to make, unfortunately: is the "date" datatype a domain, or is it just a representation of an integer or a string? Worse yet, is "I" a text string, a character, the ASCII representation of an integer value, or is it a roman numeral? There is no single correct answer, but at least the choice of datatype is simplified by moving many storage representation choices away from central application-specific code. Representation is a matter of implementation, and should not affect the core functionality of an application.

4. Run-time data manipulation

This term may seem obvious: don't all applications manipulate data at run-time?

YES
: Most applications accept information and allow users to manipulate information stored in "documents".

NO
: A wide range of actions are not dealt with as manipulating data. For example, is a click on a radio button a command to alter the state or is it equivalent to changing a variable?

Languages such as C++ create a distinction between data and actions with a heavy emphasis on "procedural code". Borland's "Delphi" programming environment demonstrates how state changes can be made to look much more like changes in object property values: as soon as a program sets the "top" property of a window object, it moves on the screen; when the user drags the window, its "top" property changes accordingly.

One example of the inconvenience caused by a pure procedural approach is selecting the sort order of some list of data displayed on the screen: the user thinks of "defining" the ordered set of fields to use for ordering all entries in the list, while the program(mer) responds by encoding this request as an array of field codes, and then using a custom sort routine (or custom comparison routine) to interpret these codes. Why can't we simply sort a list by an "ordered set of fields"?

A MORE ELABORATE EXAMPLE

Another example is factoring out common code to access data: when data is obtained from several different sources (text files, binary files, database tables, application resources, ini-files/preferences, network sites) the different origins often show up as different interfaces.

Try showing lists of documents, databases, sites, windows, drives, app resources on the screen. I bet you'll use some sort of list control with strings in it (why? because strings are a convenient domain to convert to for all sorts of data).

Lets add some spice: try that again with lists showing more specific details for each item, perhaps using a few font variations or graphics. Now, you'll probably resort to tricks like "owner-drawn controls", or derive a set of classes to override some virtual "DrawItem" member function. Good. But... how many classes or special member functions did you have to add for this "simple" change? Why can't we use general code which takes a list of fields and fonts as parameter?

This example shows how seemingly trivial issues can require a lot of administrative code to be fully resolved. A list of strings? Easy. Anything more complex, a grid perhaps? Good luck, you're on your own. Even today's "data-aware controls" are limited in many respects.

DATA-DRIVEN VS. PROCEDURAL

Objects have the property that they can easily be passed around, but to create new behavior as an object, you need to write a new class. Member functions are a bit more difficult to pass around (although templates and STL's function objects simplify this problem), but they are more easily defined.

The advantage of a run-time data representation, with rows and properties as ordinary objects, is that more programming issues can be dealt with in an application-independent manner. Concepts such as sorting, selection of a subset, counting, and conversion of datatypes, can be implemented once for (re-) use in a wide range of applications.

Content-independent utility programs for statistics, performance tuning, data import/export, etc. can now be written entirely in C++.

This is the same bright prospect that many general database packages offer: a lot of built-in functionality, just add application specifics. With the MetaKit library, you do not have to compromise: use as much or as little of it as you like, it will work fine in combination with any other library or framework. MetaKit fits in, it does not take over.

ASP Ombudsman statement

Meta Four Software is a member of the Association of Shareware Professionals (ASP). ASP wants to make sure that the shareware principle works for you. If you are unable to resolve a shareware-related problem with an ASP member by contacting the member directly, ASP may be able to help. The ASP Ombudsman can help you resolve a dispute or problem with an ASP member, but does not provide technical support for members' products. Please write to the ASP Ombudsman at 545 Grover Road, Muskegon, MI 49442-9427 USA, FAX 616-788-2765 or send a CompuServe message via CompuServe Mail to ASP Ombudsman 70007,3536.

Class members

c4_FloatProp::c4_FloatProp
c4_IntProp::c4_IntProp
c4_Row::c4_Row
c4_Row::ConcatRow
c4_Row::operator=
c4_Row::~c4_Row
c4_Storage::Commit
c4_Storage::Contents
c4_Storage::Get
c4_Storage::LoadFromStream
c4_Storage::Rollback
c4_Storage::SaveToStream
c4_Storage::Set
c4_Storage::View
c4_StringProp::c4_StringProp
c4_View::Add
c4_View::c4_View
c4_View::ElementAt
c4_View::Find
c4_View::FindProperty
c4_View::GetAt
c4_View::GetIndexOf
c4_View::GetSize
c4_View::GetUpperBound
c4_View::InsertAt
c4_View::NthProperty
c4_View::NumProperties
c4_View::operator=
c4_View::operator[]
c4_View::Project
c4_View::ProjectWithout
c4_View::RemoveAll
c4_View::RemoveAt
c4_View::Search
c4_View::Select
c4_View::SelectRange
c4_View::SetAt
c4_View::SetAtGrow
c4_View::SetSize
c4_View::Sort
c4_View::SortOn
c4_View::SortOnReverse
c4_View::~c4_View
c4_ViewProp::c4_ViewProp

Class reference

Many of the classes listed below are only used internally. If you are looking at this for the first time, you should start with the classes contained in the "k4view.h" file: the core classes are c4_View, c4_Row, and c4_Storage, with additional details in c4_Cursor and c4_RowRef. You should also look at c4_Property and the classes derived from it, which define a couple of basic datatypes.

If you want to understand the implementation: the c4_Sequence container is an essential base class of the library, as are the c4_Handler data representation handlers and the c4_Persist persistent storage manager.

c4_Allocator
c4_Bytes
c4_Column
c4_Cursor
c4_Dependencies
c4_DerivedSeq
c4_Field
c4_FilterSeq
c4_FloatProp
c4_FloatRef
c4_Handler
c4_HandlerSeq
c4_IntProp
c4_IntRef
c4_Notifier
c4_Persist
c4_ProjectSeq
c4_Property
c4_Reference
c4_Row
c4_RowRef
c4_Sequence
c4_SortSeq
c4_Storage
c4_Streamer
c4_StringProp
c4_StringRef
c4_Table
c4_TableEntry
c4_View
c4_ViewProp
c4_ViewRef

Contact information

```
Jean-Claude Wippler          mailto:jcw@meta4.nl

Meta Four Software             http://purl.net/meta4
Meekrap oord 6
3991 VE, Houten                 fax:+31 30 635 2337
The Netherlands
```

Description

```
The MetaKit Library is a software development library for data storage
and easy manipulation of structured objects and collections in C++.
If you need bulky, powerful database functions look elsewhere, but if you
need a small self-contained library to store structured data, read on.

This software is distributed as shareware. Full source code is available.
Please read the disclaimer, which is in the file "include\m4kit.h".

For the latest news about MetaKit, point your Internet browser to:
                         http://purl.net/meta4/metakit

For an example of its use, check out the "CatFish" disk catalog browser.
This freeware Windows utility demonstrates the size and speed of MetaKit.
```

Distribution and support

The MetaKit Library software package is copyright Meta Four Software, NL. You may not distribute this information, nor any portions thereof, as part of a general database- or object-persistence software library or utility.

You may distribute this shareware release for any other purpose, provided that all files are included without change, including this description.

Meta Four Software will provide technical support for 12 months, starting on the day your registration fee is received. If a problem with MetaKit is not resolved according to the specifications, you will be given the option to receive a refund of the purchase price.

Functions

operator

Getting started

To install this library and sample code, unpack the ZIP archive to a new directory (such as "C:\M4KIT"). Make sure to keep the original directory structure, with 'Use Directory Names' in WinZip or 'pkunzip -d' in MS-DOS.

Please read the "WHATSNEW.TXT" file for the last-minute release details.

The main introduction is in the file "M4KIT.HTM", which is in HTML format. A copy of this is in "M4KIT.WRI", for use with Write/WordPad/QuickView. The file "M4KITAPI.HLP" is the reference guide for this library (Winhelp).

A simple disk-catalog program is included to illustrate many features. Emphasis is on data-manipulation, this example uses a rudimentary UI. The source code is in "examples\discat", the 16-bit Windows executable is called "bin\discat16.exe", the 32-bit version is "bin\discat.exe".

Note: You will need Microsoft's MFC250.DLL to run the 16-bit version, or the two files MFC40.DLL & MSVCRT40.DLL to run the 32-bit version. If you do not have them, you may not be able to do much with the current MFC-only version of this software library, unfortunately.

The "dump.exe" utility dumps the contents of datafiles, the "struct.exe" program shows their data structure, full source code for both is included. The "catsend.exe" and "catrecv.exe" sample applications demonstrate the exchange of MetaKit data over TCP/IP. The "ftpcat.exe" 32-bit console application creates catalog of remote FTP server directories (which are compatible with "discat").

The CatFish disk catalog browser executable is available separately (see the MetaKit home page), the source and makefile for a DLL version of this program are included in this distribution.

Overview

Portability

This release has been tested with Microsoft Visual C++ 4.0 & 4.1 (Win 32), 1.52 (Win 16 and MSDOS), Symantec C++ 7.2 (Win 32), and Watcom C++ 10.5 (Win 32) and uses files/strings/containers of MFC (Microsoft Foundation Classes framework). Only MFC-based development is currently supported. The library can be linked statically, as user DLL, or as an extension DLL.

Some newer C++ features such as templates and RTTI are not required for this release (exceptions are used, using MFC macros for 16-bit MSVC 1.52). The generic data types can be implemented surprisingly well without them, although templates (and STL) will be used once they are widely supported.

All C++ identifiers with global scope include the digit four. Namespace support will be added when this addition to C++ becomes more widespread.

Support for other compilers, frameworks, hardware platforms, and operating systems will be included in upcoming 1996 releases. But... in which order? Borland C++, Mac/CodeWarrior, STL, Unix, anything else? Help! Let me know!

Purpose of MetaKit

```
MAKE YOUR STRUCTURED DATA PERSISTENT
Data is loaded on-demand, with immediate access to any size data files.
Modifications are only made permanent when explicitly committed.

GET THE ADVANTAGES OF A DATABASE IN C++
Any failure, even during commit, causes automatic rollback on next open.
Automatic file storage allocation, all repair/reclaim is automatic.

TRANSPORT YOUR DATA ANYWHERE
All data can be flattened for streaming over a communications channel.
Files are portable across platforms, several library ports are planned.

RUN-TIME DATA STRUCTURES
Data structures are dynamic, allowing you to write generic utilities.
Deal with file format changes and multiple formats as your code evolves.

HIGH PERFORMANCE
Can store millions of objects, instant access to any view/row/property.
Storage overhead for large collections of small objects can be very low.

NO INCONVENIENT COMPROMISES
MetaKit does not depend on UI code, so it can be combined with any GUI.
Existing class libraries are used for string & container implementation.

EASY TO READ, EASY TO USE
Only a few tiny header files, distracting details are well encapsulated.
Uses view / row / property classes with [] and () operator overloading.

YOU STAY IN CONTROL
The code footprint of MetaKit is small, you can link it as LIB or DLL.
Complete, commented source code of MetaKit is available. No royalties.

ONE SIZE FITS ALL
Small projects: simple headers, easy to work with, small code, any GUI.
Very large projects: complex structures, huge files, portable, failsafe.

SOLID AND SUPPORTED
This design has evolved during many years of commercial C++ programming.
Your questions, comments, and suggestions will be taken very seriously.
```

Registration

The MetaKit Library is not free, you must register for US$ 25 if you use this code in your own software after an evaluation period of 30 days. When used commercially, you must register and purchase the source code license for another US$ 65 to obtain the right to incorporate this library code.

Payment of the US$ 25 registration fee entitles you to:

The right to use this software for personal and non-profit use
Obtain all the most recent library versions (debug & release)
Timely notification by email of any new revisions and updates
Free registration of every revision for the coming 12 months
Quick response times for all technical support questions
A warm thank you - you're supporting me to continue this effort!

By purchasing the source code license for an additional US$ 65, you get:

The right to include and distribute object code for commercial use
The complete, commented source code in C++ of the MetaKit library
A twelve month subscription to all major updates via email

How to register your copy or purchase the source code version:

CREDIT CARD, CASH, CHECK, OR MONEY ORDERS:
You can use the enclosed REGISTER.EXE program to send your order to the KAGI shareware registration service, using email, fax, or regular mail.

COMPUSERVE: GO SWREG, select reg# 10351 (10352 for source code), and enter all details. Confirmation and any further information will then be sent to you by email within two business days.

PHONE ORDERS: You can call PsL at 800-2424-PsL or 713-524-6394 and give them a MC, Visa, Amex, or Discover card number. Specify item #14522 and whether you want source code. To insure that you will receive the latest registered version, PsL will notify us the day of your order and additional information will be sent to you within two business days.

NOTE: THE ABOVE PHONE NUMBERS ARE FOR CREDIT CARD, ETC. ORDERS ONLY. THE AUTHOR OF THIS LIBRARY CANNOT BE REACHED AT THESE NUMBERS.

Any questions about the status of the shipment of the order, refunds, registration options, product details, tech support, volume discounts, site licenses, etc, must be directed to the email/fax addresses below.

Roadmap

Several header files and libraries have the digit four in their name to avoid potential name conflicts with other files on your system.

DIRECTORIES

The shareware distribution has the following directory structure:

```
bin\        Executables of the examples
examples\   Source code examples
include\    Header files
lib\        Libraries needed to build applications
```

These directories contain everything you need to build applications for 32-bit and 16-bit Windows, as well as for MS-DOS (large-model).

In addition, the source code version includes these directories:

```
src\        C++ source files and private headers
msvc152\    Microsoft Visual C++ 1.52 build areas
msvc41\     Microsoft Visual C++ 4.1 build areas
```

HEADER FILES

The MetaKit library software is highly modular (loosely coupled). As a consequence, the headers you need to look at are quite small. The following headers are provided in the shareware version:

```
include\m4kit.h     The single header you'll need to include.
```

This is a wrapper for the following files:

```
include\k4conf.h    Compatibility and common definitions
include\k4view.h    Definition of views, cursors, and rows
```

The remaining include files contain some less essential details:

```
include\k4table.h   Persistent storage details (low-level)
include\k4field.h   Structured field details (low-level)
include\k4viewx.h   Additional view classes, used internally
include\k4*.inl     Inline function definitions
```

These files define all capabilities of the library, no more, no less. The core functionality is provided by the View and Storage classes.

SOURCE CODE EXAMPLES

These programs illustrate different aspects of the MetaKit library:

```
examples\discat     A basic disk catalog, for Win16 and Win32.
```

```
                         Compiled as a small dialog-based application.
                         Shows use with a non-trivial data structure.

    examples\dump       A utility to dump the contents of a datafile.
                         Compiled as a (32-bit) console application.
                         Useful for debugging, various output formats.

    examples\struct     This utility displays the data structure of a
                         datafile as a graph. It is distributed as an
                         MS-DOS large-model real-mode executable.

    examples\catsend    Sends catalog file over a TCP/IP connection.
                         Compiled with MSVC 4 (Win32), uses Winsock.
                         Requires a catalog file created with discat.

    examples\catrecv    Receives and displays disk catalog as a tree.
                         In MSVC 4.1, requires Win95 for the new Tree
                         control (or perhaps Win32s version 1.30).
                         Requires catsend to feed it some information.

    examples\ftpcat     Similar to discat, but this console app will
                         generate tree catalogs of remote ftp servers.
                         Its catalog files are compatible with catsend.
                         Requires MSVC 4.x, runs FTP.EXE for sessions.

    examples\catfish    Source code of the freeware "CatFish" utility,
                         a tiny but very fast disk catalog browser.
                         Created with MSVC 1.52 as a Win16 application.
                         Does not require, but works fine with FTPCAT.
```

The CATSEND/CATRECV programs work together to demonstrate streaming, generic file access (CATSEND knows nothing about the DISCAT format), and use without data files (CATRECV only captures and displays data). You need WINSOCK and TCP/IP, but you can run them on a single machine. The FTPCAT utility creates files compatible with CATSEND and CATRECV. Finally, CATFISH is a full-scale utility application, with source.

Source files

CONF.H - Public configuration header
CONFIG.H - Wrapper for the compatibility headers
DEFS.H - Set up the "q4_*" defines
DERIVED.CPP - Derived sequence implementation.
FIELD.CPP - Field implementation.
FIELD.H - Field declarations
FIELD.INL - Field inline declarations
FIX.H - Header to fix compatibility problems
FORMAT.CPP - Format handler implementation.
HANDLER.CPP - Data handler implementation.
HEADER.H - First header in sources
MFC.H - Microsoft Foundation Classes
MSVC.H - Microsoft Visual C++
PERSIST.CPP - Persistence implementation.
PERSIST.H - Persistence declarations
STORE.CPP - Storage implementation.
STORE.H - Storage declarations
TABLE.CPP - Table implementation.
TABLE.H - Table declarations
TABLE.INL - Table inline declarations
TAIL.H - Closing definitions
VIEW.CPP - View implementation.
VIEW.H - View declarations
VIEW.H - View inline declarations
VIEWX.CPP - Auxiliary View implementation.
VIEWX.H - Auxiliary view declarations

Structures and Enums

Module CONF.H - Public configuration header

Filename: C:/JC/SRC/MKW/INCLUDE/K4CONF.H

Description

This is the simplified header for use with the MetaKit library.

There is no point in exporting the bulky and complex conditional header structure used within the library itself, but the definitions below will have to be changed substantially to support new compilers, etc.

Developer Notes

Version 1.2, jcw@meta4.nl

Module CONFIG.H - Wrapper for the compatibility headers

Filename: C:/JC/SRC/MKW/SRC/CONFIG.H

Description
This file wraps all machine dependencies and compiler dependent stuff.

Developer Notes
Version 1.2, jcw@meta4.nl

Module DEFS.H - Set up the "q4_*" defines

Filename: C:/JC/SRC/MKW/SRC/DEFS.H

Description
The task of this file is to figure out what each compiler environment is trying to tell us through its predefined constants, either from the pre- processor or from the IDE. Configuration choices should go in "fix.h".

Developer Notes
Version 1.2, jcw@meta4.nl

Module DERIVED.CPP - Derived sequence implementation.

Filename: C:/JC/SRC/MKW/SRC/DERIVED.CPP

Description
This file contains the implementation of the derived sequences.

To Do
implement propagation of sort, solve a few complex problems

make sorting thread-safe, get rid of globals (STL sort fixes it)

simplify this stuff, it's becoming too large...

Developer Notes
Version 1.2, jcw@meta4.nl

Module FIELD.CPP - Field implementation.

Filename: C:/JC/SRC/MKW/SRC/FIELD.CPP

Description
This file contains the implementation of the field classes.

Developer Notes
Version 1.2, jcw@meta4.nl

FIELD.CPP - Field implementation.
FIELD.H - Field declarations
FIELD.INL - Field inline declarations
c4_Field

Module FIELD.H - Field declarations

Filename: C:/JC/SRC/MKW/INCLUDE/K4FIELD.H

Description

This file contains the declaration of the field classes.

Developer Notes

Version 1.2, jcw@meta4.nl

Module FIELD.INL - Field inline declarations

Filename: C:/JC/SRC/MKW/INCLUDE/K4FIELD.INL

Description
This file contains inline definitions for the field classes.

Only very small members are selected for inline code. See the class listing for a description of these functions.

Developer Notes
Version 1.2, jcw@meta4.nl

Module FIX.H - Header to fix compatibility problems

Filename: C:/JC/SRC/MKW/SRC/FIX.H

Description

This is the place to alter the defines which were set up in "defs.h". If necessary, code from "config.h" can be duplicated and adjusted here. By placing all fixes here, future updates will be far easier to install. See "config.h" for further comments on how to use this file.

Developer Notes

Version 1.2, jcw@meta4.nl

Module FORMAT.CPP - Format handler implementation.

Filename: C:/JC/SRC/MKW/SRC/FORMAT.CPP

Description
This file contains the implementation of the format handler classes.

Developer Notes
Version 1.2, jcw@meta4.nl

FORMAT.CPP - Format handler implementation.

Module HANDLER.CPP - Data handler implementation.

Filename: C:/JC/SRC/MKW/SRC/HANDLER.CPP

Description
This file contains the implementation of the data handler classes.

Developer Notes
Version 1.2, jcw@meta4.nl

HANDLER.CPP - Data handler implementation.

Module HEADER.H - First header in sources

Filename: C:/JC/SRC/MKW/SRC/HEADER.H

Description
This header file is always the first in each library source file.

Developer Notes
Version 1.2, jcw@meta4.nl

Module MFC.H - Microsoft Foundation Classes

Filename: C:/JC/SRC/MKW/SRC/MFC.H

Description

Specific header for the Microsoft Foundation Classes.

Developer Notes

Version 1.2, jcw@meta4.nl

Module MSVC.H - Microsoft Visual C++

Filename: C:/JC/SRC/MKW/SRC/MSVC.H

Description

Specific header for the Microsoft Visual C++ development tools.

Developer Notes

Version 1.2, jcw@meta4.nl

Module PERSIST.CPP - Persistence implementation.

Filename: C:/JC/SRC/MKW/SRC/PERSIST.CPP

Description
This file contains the implementation of the persistence classes.

To Do
Use chunky stream instead of MFC's archive.

Developer Notes
Version 1.2, jcw@meta4.nl

PERSIST.CPP - Persistence implementation.
PERSIST.H - Persistence declarations
c4_Allocator
c4_Persist
c4_Streamer
c4_TableEntry

Module PERSIST.H - Persistence declarations

Filename: C:/JC/SRC/MKW/SRC/PERSIST.H

Description

This file contains the declaration of the persistence classes.

Developer Notes

Version 1.2, jcw@meta4.nl

Module STORE.CPP - Storage implementation.

Filename: C:/JC/SRC/MKW/SRC/STORE.CPP

Description
This file contains the implementation of the storage classes.

To Do
automatically generate description strings

Developer Notes
Version 1.2, jcw@meta4.nl

STORE.CPP - Storage implementation.
STORE.H - Storage declarations
c4_Dependencies
c4_DerivedSeq
c4_Handler
c4_HandlerSeq
c4_Notifier
c4_Storage::Commit
c4_Storage::Contents
c4_Storage::LoadFromStream
c4_Storage::Rollback
c4_Storage::SaveToStream

Module STORE.H - Storage declarations

Filename: C:/JC/SRC/MKW/SRC/STORE.H

Description

This file contains the declaration of the storage classes.

To Do

Updatable views (where appropriate, lots of tricky details).

Developer Notes

Version 1.2, jcw@meta4.nl

Module TABLE.CPP - Table implementation.

Filename: C:/JC/SRC/MKW/SRC/TABLE.CPP

Description

This file contains the implementation of the table classes.

To Do

Don't load data now from flat format if not necessary.

Commit over flat format is destructive, shouln't be.

Developer Notes

Version 1.2, jcw@meta4.nl

TABLE.CPP - Table implementation.
TABLE.H - Table declarations
TABLE.INL - Table inline declarations
c4_Bytes
c4_Column
c4_Table

Module TABLE.H - Table declarations

Filename: C:/JC/SRC/MKW/INCLUDE/K4TABLE.H

Description
This file contains the declaration of the table classes.

Developer Notes
Version 1.2, jcw@meta4.nl

Module TABLE.INL - Table inline declarations

Filename: C:/JC/SRC/MKW/INCLUDE/K4TABLE.INL

Description

This file contains inline definitions for the table classes.

Only very small members are selected for inline code. See the class listing for a description of these functions.

Developer Notes

Version 1.2, jcw@meta4.nl

Module TAIL.H - Closing definitions

Filename: C:/JC/SRC/MKW/SRC/TAIL.H

Description

This header adds any remaining defines, typedefs, classes, etc.

Developer Notes

Version 1.2, jcw@meta4.nl

Module VIEW.CPP - View implementation.

Filename: C:/JC/SRC/MKW/SRC/VIEW.CPP

Description
This file contains the implementation of the main view classes.

Views act as a general interface to collections of data objects. Generic "rows" of data can be added, modified, and removed from a view almost like a normal C++ object with a C++ container class.

A major difference between these views and static C++ objects, is that the structure of views can be defined and altered at run time. Access to entries in views is provided by instances of the class c4_Row. Access to the data items in rows is managed by instances of c4_Property. The views themselves are encapsulated by the class c4_View. A reference-counting scheme is used to track the real workhorses of this mechanism, the classes derived from c4_Sequence.

To Do
use binary search for c4_View::Search

Developer Notes
Version 1.2, jcw@meta4.nl

Module VIEW.H - View declarations

Filename: C:/JC/SRC/MKW/INCLUDE/K4VIEW.H

Description
This file contains the declaration of the main view classes.

Developer Notes
Version 1.2, jcw@meta4.nl

Module VIEW.H - View inline declarations

Filename: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Description
This file contains inline definitions for the view classes.

Only very small members are selected for inline code. See the class listing for a description of these functions.

Developer Notes
Version 1.2, jcw@meta4.nl

Module VIEWX.CPP - Auxiliary View implementation.

Filename: C:/JC/SRC/MKW/SRC/VIEWX.CPP

Description
This file contains the implementation of the auxiliary view classes.

Developer Notes
Version 1.2, jcw@meta4.nl

VIEWX.CPP - Auxiliary View implementation.
VIEWX.H - Auxiliary view declarations
c4_FloatRef
c4_IntRef
c4_Reference
c4_Sequence
c4_StringRef
c4_ViewRef

Module VIEWX.H - Auxiliary view declarations

Filename: C:/JC/SRC/MKW/INCLUDE/K4VIEWX.H

Description
This file contains the declaration of the auxiliary view classes.

Developer Notes
Version 1.2, jcw@meta4.nl

c4_Allocator Class

class c4_Allocator

Support class to track allocated storage in an abstract container.

Defined in: C:/JC/SRC/MKW/SRC/PERSIST.H

c4_Bytes Class

class c4_Bytes

These fellows know how to clean up their act when they go away.

These objects are used to pass around untyped data without concerns about clean-up. They know whether the bytes need to be de-allocated when these objects go out of scope. Small amounts of data are stored in the object.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4TABLE.H

Class Members

c4_Bytes()
Constructs an empty object.

c4_Bytes(const void*, int, bool)
Constructs an object with contents, optionally as a copy.

c4_Bytes(const c4_Bytes&)
Copy constructor.

~c4_Bytes()
Destructor.

c4_Bytes& operator=(const c4_Bytes&)
Assignment, this may make a private copy of contents.

void Swap(c4_Bytes&)
Swaps the contents and ownership of two byte objects.

int Size() const
Returns the number of bytes of its contents.

const uchar* Contents() const
Returns a pointer to the contents.

uchar* SetBuffer(int)
Defines contents as a freshly allocated buffer of given size.

uchar* SetBufferClear(int)
Allocates a buffer and fills its contents with zero bytes.

c4_Column Class

class c4_Column

Objects of this class define the column contents of tables.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4TABLE.H

See Also
c4_Table

Class Members
Public members

uchar* Contents() const
Provides access to the contents if loaded.

void SetPointer(uchar*, int)
Sets the buffer pointer and the allocated size.

uchar* SetBuffer(int)
Allocate a new buffer of the specified size.

uchar* SetBufferClear(int)
Allocate and clear a new buffer of the specified size.

ulong Size() const
Returns the number of bytes as stored on disk.

void SetDirty()
Sets a flag to save this column on commit.

bool IsDirty() const
Returns true if contents needs to be saved.

uchar* LoadNow(c4_Persist&)
Make sure the data is loaded into memory.

bool SaveNow(c4_Streamer&)
Save the buffer if dirty, also writes header.

c4_Cursor Class

class c4_Cursor

A wrapper class of objects to iterate over sequences of rows.

Cursor objects can be used to point to specific entries in a view. A cursor acts very much like a pointer to a row in a view, and is returned when taking the address of a c4_RowRef. Dereferencing a cursor leads to the original row reference again. You can construct a cursor for a c4_Row, but since such rows are not part of a collection, incrementing or decrementing these cursors is meaningless (and wrong).

The usual range of pointer operations can be applied to these objects: pre/post-increment and decrement, adding or subtracting integer offsets, as well as with the full range of comparison operators. If two cursors point to entries in the same view, their difference can be calculated.

As with regular pointers, care must be taken to avoid running off of either end of a view (the debug build includes assertions to check this).

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.H

Class Members
Public members

Construction / destruction / dereferencing

c4_Cursor(c4_Sequence&, int)
Constructs a new cursor.

c4_RowRef operator*() const
Dereferences this cursor to "almost" a row.

c4_RowRef operator[](int) const
This is the same as *(cursor + offset).

Stepping the iterator forwards / backwards

c4_Cursor& operator++()
Pre-increments the cursor.

c4_Cursor operator++(int)
Post-increments the cursor.

c4_Cursor& operator--()
Pre-decrements the cursor.

c4_Cursor operator--(int)
Post-decrements the cursor.

c4_Cursor& operator+=(int)
Advances by a given offset.

c4_Cursor& operator-=(int)
Backs up by a given offset.

c4_Cursor operator-(int) const
Subtracts a specified offset.

int operator-(c4_Cursor) const
Returns the distance between two cursors.

friend c4_Cursor operator+(c4_Cursor, int)
Adds specified offset.

friend c4_Cursor operator+(int, c4_Cursor)

Adds specified offset to cursor.

Comparing row positions

friend bool operator==(c4_Cursor, c4_Cursor)
Returns true if both cursors are equal.

friend bool operator!=(c4_Cursor, c4_Cursor)
Returns true if both cursors are not equal.

friend bool operator<(c4_Cursor, c4_Cursor)
True if first cursor is less than second cursor.

friend bool operator>(c4_Cursor, c4_Cursor)
True if first cursor is greater than second cursor.

friend bool operator<=(c4_Cursor, c4_Cursor)
True if first cursor is less or equal to second cursor.

friend bool operator>=(c4_Cursor, c4_Cursor)
True if first cursor is greater or equal to second cursor.

c4_Dependencies Class

class c4_Dependencies

A dependencies object keeps track of the set of dependent views.

Defined in: C:/JC/SRC/MKW/SRC/STORE.H

c4_DerivedSeq Class

class c4_DerivedSeq: public c4_Sequence

A derived sequence delegates most of its work to another sequence.

This is an abstract base class, which is used by several other classes including c4_SortSeq, c4_FilterSeq, and c4_ProjectSeq. The basic mechanism here is to store a pointer to an underlying sequence and to funnel all requests through to it. Derived classes can adjust this behavior to suit their purposes. Note that derived sequences fall into roughly two categories: row mapping and property mapping. Row mapping modifies the row index before calling the underlying sequence, while property mapping alters the use of properties or data handlers.

Defined in: C:/JC/SRC/MKW/SRC/STORE.H

c4_Field Class

class c4_Field

Fields form a tree describing the underlying data structure.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4FIELD.H

Class Members
Public members

Construction / destruction

c4_Field(const char*&, c4_Field* =0)
Constructs a new field.

~c4_Field()
Destructor.

Structural information

bool IsRoot() const
Returns true if this is the root field.

c4_Field& Parent() const
returns the field which contains this one.

int Index() const
Returns the index of this field in its parent.

int Level() const
Returns the nesting level of this field.

int Degree() const
Returns the repetition degree of this field.

c4_Field& FindContainer()
Returns the enclosing repeating field.

Repeating and compound fields

int NumSubFields() const
Returns the number of subfields.

c4_Field& SubField(int) const
Returns the description of each subfield.

bool IsRepeating() const
Returns true if this field contains subtables.

int NumSubColumns() const
Returns total number of columns in the subtables.

c4_Field& SubColumn(int) const
Returns the description of each subtable column.

Field name and description

c4_String Name(bool =false) const
Returns name of this field, default omits details.

c4_String Type() const
Returns the type description of this field, if any.

c4_String Tag() const
Returns the fully specified name of this field.

c4_String Description(bool =false) const
Describes the structure, can omit field names.

int FindSubField(const char*) const
Finds the subfield index by name.

Column offsets and counts

int Offset() const
Returns start of the first column in the table.

int Width() const
Returns number of columns used in the table.

int Depth() const
Returns depth to the table containing the data.

int FindIndex(int) const
Finds the field index of a given table column.

c4_FilterSeq Class

class c4_FilterSeq: public c4_DerivedSeq

A filtering sequence is a derived sequence which does row selection.

An array of index values is built and saved during construction. This array determines which entries of the underlying sequence are accepted.

Defined in: C:/JC/SRC/MKW/SRC/DERIVED.CPP

c4_FloatProp Class

class c4_FloatProp: public c4_Property

Class of objects representing floating point properties.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.H

Class Members
Public members

c4_FloatProp**(const char*)**
    Constructs a new property.

c4_Row operator[](double)
    Creates a row with a single floating point value.

c4_FloatRef operator()(c4_RowRef)
    Gets or sets an floating point property of a row.

c4_FloatProp::c4_FloatProp

d4_inline c4_FloatProp::c4_FloatProp(const char* ***name_*****)**

Constructs a new floating point property.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Parameters

*name_*
  the name of this property

c4_FloatRef Class

class c4_FloatRef: public c4_Reference

Used to get or set floating point values.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEWX.H

Class Members
Public members

c4_FloatRef(const c4_Reference&)
Constructor.

c4_Handler Class

class c4_Handler

The data handler deals with issues concerning data representation.

Defined in: C:/JC/SRC/MKW/SRC/STORE.H

Class Members
Public members

c4_Handler()
Constructor.

virtual ~c4_Handler()
Destructor.

virtual void Initialize(c4_HandlerSeq&, int, c4_Property&)
Initializes the data handler once context is known.

virtual void Defined()
Called when the corresponding table has been fully defined.

int PropId() const
Returns the id of the property associated to this handler.

virtual void Clear(c4_Bytes& buf_)
Returns the default value for items of this type.

virtual void Get(int index_, c4_Bytes& buf_)
Retrieves the data item at the specified index.

virtual void Set(int index_, const c4_Bytes& buf_)
Stores a new data item at the specified index.

virtual int Compare(int index_, const c4_Bytes& buf_)
Compares an entry with a specified data item.

virtual void Insert(int index_, const c4_Bytes& buf_, int count_)
Inserts 1 or more data items at the specified index.

virtual void Remove(int index_, int count_)
Removes 1 or more data items at the specified index.

void Move(int from_, int to_)
Move a data item to another position.

c4_HandlerSeq Class

class c4_HandlerSeq

A handler sequence uses data handlers for actual access to the data.

This is a type of sequence which actually stores data items (as opposed to derived sequences, which rely on other sequences to deal with this). Information is managed by data handlers, which are usually organized to store information as column-wise vectors.

Defined in: C:/JC/SRC/MKW/SRC/STORE.H

c4_IntProp Class

class c4_IntProp: public c4_Property

Class of objects representing integer properties.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.H

Class Members
Public members

c4_IntProp**(const char*)**
Constructs a new property.

c4_Row operator[](long)
Creates a row with a single integer.

c4_IntRef operator()(c4_RowRef)
Gets or sets an integer property of a row.

c4_IntProp::c4_IntProp

d4_inline c4_IntProp::c4_IntProp(const char* ***name_*****)**

Constructs a new integer property.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Parameters

*name_*
: the name of this property

c4_IntRef Class

class c4_IntRef: public c4_Reference

Used to get or set integer values.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEWX.H

Class Members
Public members

c4_IntRef(const c4_Reference&)
Constructor.

c4_Notifier Class

class c4_Notifier

A notifier manages the process of notifying all derived views.

Notification is a very powerful (and complex) mechanism which is used to propagate changes in views to all derived views. The complexity is caused by the fact that changes will cause views to be in an inconsistent state during change propagation. Pre- and post-changes are used to handle this.

Defined in: C:/JC/SRC/MKW/SRC/STORE.H

c4_Persist Class

class c4_Persist

This object manages persistent storage of tables on file.

Defined in: C:/JC/SRC/MKW/SRC/PERSIST.H

Class Members

Public members

c4_Persist(c4_File* =0, c4_Table* =0)
Constructs a new object.

~c4_Persist()
Destructor.

c4_Table* Root() const
Returns the root table of this object.

c4_File* File() const
Returns the file associated with this object.

bool Commit()
Flushes pending changes to file right now.

bool Rollback()
(Re)initializes for on-demand loading. This will cancel any uncommitted changes.

bool Load(c4_File&)
Loads contents from the specified input stream.

bool Save(c4_File&)
Saves contents to the specified output stream.

c4_ProjectSeq Class

class c4_ProjectSeq: public c4_DerivedSeq

A projecting sequence is a derived sequence which rearranges fields.

Defined in: C:/JC/SRC/MKW/SRC/DERIVED.CPP

c4_Property Class

class c4_Property

Properties correspond to the subfields of data rows.

Property objects exist independently of view, row, and storage objects. They have a name and type, and can be used by multiple views. You will normally only use derived classes for strong typing.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.H

See Also
c4_IntProp, c4_FloatProp, c4_StringProp, c4_ViewProp

Class Members
Public members

c4_Property(char, const char*)
Constructs a new property with the give type and name.

c4_Property(int)
Constructs a property from its id.

~c4_Property()
Destructor.

c4_String Name() const
Returns the name of this property.

char Type() const
Returns the type of this property.

int GetId() const
Returns a unique id for this property. Properties with the same type and name always share the same id.

c4_Reference operator()(c4_RowRef) const
Gets or sets this untyped property in a row.

void Refs(int) const
Adjusts the reference count.

static void CleanupCache()
Removes unused property info and deallocates the cache.

c4_Reference Class

class c4_Reference

A reference is used to get or set typed data, using derived classes.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEWX.H

See Also
c4_IntRef, c4_FloatRef, c4_StringRef, c4_ViewRef

Class Members
Public members

c4_Reference(c4_RowRef, int)
Constructor.

c4_Reference& operator=(const c4_Reference&)
Assignment of one data item.

bool GetData(c4_Bytes&) const
Retrieves the value of the referenced data item.

void SetData(const c4_Bytes&) const
Stores a value into the referenced data item.

c4_Row Class

class c4_Row: public c4_RowRef

A row is an entry in a sequence with copy semantics. Rows can exist by themselves and as the contents of views. Row assignment implies that a copy of the contents of the originating row is made.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.H

Class Members
Public members

c4_Row**()**
  Constructs a row with no properties.

c4_Row**(const c4_Row&)**
  Constructs a row from another one.

c4_Row**(const c4_RowRef&)**
  Constructs a row copy from a row reference.

~c4_Row**()**
  Destructor.

c4_Row& operator=**(const c4_Row&)**
  Assigns a copy of another row to this one.

c4_Row& operator=**(const c4_RowRef&)**
  Copies another row to this one.

void ConcatRow**(const c4_RowRef&)**
  Adds all properties and values into this row.

friend c4_Row operator+(const c4_RowRef&, const c4_RowRef&)
  Returns a new row which is the concatenation of two others.

c4_Row::c4_Row

c4_Row::c4_Row(void)

Default constructor.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

c4_Row::ConcatRow

void c4_Row::ConcatRow(const c4_RowRef& ***rowRef_*****)**

Adds all properties and values into this row.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Parameters

*rowRef_*
    the row to concatenate from

c4_Row::operator=

c4_Row& c4_Row::operator=(const c4_RowRef& ***rowRef_*****)**

Assignment from a reference to a row.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Parameters
*rowRef_*
the row reference to assign from

c4_Row::~c4_Row

c4_Row::~c4_Row(void)

Destructor.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

c4_RowRef Class

class c4_RowRef

Row references can be used on either side of an assignment.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.H

Class Members
Public members

General operations

c4_RowRef operator=(const c4_RowRef&)
Assigns the value of another row to this one.

c4_Cursor operator&() const
Returns the cursor associated to this row.

c4_View Container() const
Returns the underlying container view.

Comparing row contents

friend bool operator==(c4_RowRef, c4_RowRef)
Returns true if the contents of both rows is equal.

friend bool operator!=(c4_RowRef, c4_RowRef)
Returns true if the contents of both rows is not equal.

friend bool operator<(c4_RowRef, c4_RowRef)
True if first row is less than second row.

friend bool operator>(c4_RowRef, c4_RowRef)
True if first row is greater than second row.

friend bool operator<=(c4_RowRef, c4_RowRef)
True if first row is less or equal to second row.

friend bool operator>=(c4_RowRef, c4_RowRef)
True if first row is greater or equal to second row.

c4_Sequence Class

class c4_Sequence

A sequence is an abstract base class for views on ranges of records.

Sequences represent arrays of rows (or indexed collections / tables). Insertion and removal of entries is allowed, but can take linear time. A reference count is maintained to decide when the object should go away.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEWX.H

Class Members
Public members

General

c4_Sequence()
Abstract constructor.

virtual int Compare(int, c4_Cursor) const
Compares the specified row with another one.

virtual void SetAt(int, c4_Cursor)
Replaces the contents of a specified row.

virtual int RemapIndex(int, const c4_Sequence*) const
Remaps the index to an underlying view.

c4_String Describe() const
Returns a descriptions of the current data structure.

Reference counting

void IncRef()
Increments the reference count of this sequence.

void DecRef()
Decrements the reference count, delete objects when last.

int NumRefs() const
Returns the current number of references to this sequence.

Adding / removing rows

virtual int Size() const
Returns the current number of rows.

void Resize(int, int =-1)
Changes number of rows, either by inserting or removing them.

virtual void InsertAt(int, c4_Cursor, int =1)
Inserts one or more rows into this sequence.

virtual void RemoveAt(int, int =1)
Removes one or more rows from this sequence.

virtual void Move(int, int)
Move a row to another position.

Properties

int NthProperty(int) const

Returns the id of the N-th property.

int PropIndex(int, bool =false)
Finds the index of a property, or creates a new entry.

virtual int NumHandlers() const
Returns the number of data handlers in this sequence.

virtual c4_Handler& NthHandler(int) const
Returns a reference to the N-th handler in this sequence.

virtual const c4_Sequence* HandlerContext(int) const
Returns the context of the N-th handler in this sequence.

virtual int AddHandler(c4_Property&, c4_Handler*)
Adds the specified data handler to this sequence.

Element access

virtual bool Get(int, int, c4_Bytes&)
Retrieves one data item from this sequence.

virtual void Set(int, int, const c4_Bytes&)
Stores a data item into this sequence.

Dependency notification

void Attach(c4_Sequence* child_)
Registers a sequence to receive change notifications.

void Detach(c4_Sequence* child_)
Unregisters a sequence which received change notifications.

c4_Dependencies* GetDependencies() const
Returns a pointer to the dependencies, or null.

virtual c4_Notifier* PreChange(c4_Notifier& nf_)
Called just before a change is made to the sequence.

virtual void PostChange(c4_Notifier& nf_)
Called after changes have been made to the sequence.

c4_SortSeq Class

class c4_SortSeq: public c4_DerivedSeq

A sorting sequence is a derived sequence which sorts its rows.

Defined in: C:/JC/SRC/MKW/SRC/DERIVED.CPP

c4_Storage Class

class c4_Storage

Objects of this class manage persistent storage of view structures.

The storage class uses a view, with additional functionality to be able to store and reload the data it contains (including nested subviews).

By default, data is loaded on demand, i.e. whenever data which has not yet been referenced is used for the first time. Loading is limited to the lifetime of this storage object, since the storage object carries the file descriptor with it that is needed to access the data file.

To save changes, call the Commit member. This is the only time that data is written to file - when using a read-only file simply avoid calling Commit.

The LoadFromStream and SaveToStream members can be used to serialize the contents of this storage row using only sequential I/O (no seeking, only read or write calls).

The data storage mechanism implementation provides fail-safe operation: if anything prevents Commit from completing its task, the last succesfully committed version of the saved data will be recovered on the next open. This includes all changes made to the table structure.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.H

Example

The following code creates a view with 1 row and stores it on file:

```
c4_StringProp pName ("Name");
c4_IntProp pAge ("Age");

c4_View myView;
myView.Add(pName ["John Williams"] + pAge [43]);

c4_Storage storage ("myfile.dat", "Musicians[Name:S,Age:I]");
storage.View("Musicians") = myView;
```

Class Members

Public members

c4_Storage(const char*, const char*)
Constructs a storage object for given file and format.

c4_Storage(const char*, bool canModify_ =false)
Constructs a storage object, keeping the current structure.

c4_Storage(c4_File* =0)
Constructs a storage object based on the given file.

~c4_Storage()
Destructor.

c4_RowRef Contents**() const**
Gives access to the stored data as a single row.

c4_File* File() const
Returns the file associated with this object.

c4_Table& RootTable() const
Returns the root table entry.

c4_String Description() const

Returns a description of the data structure.

bool Commit**()**
Flushes pending changes to file right now.

bool Rollback**()**
(Re)initializes for on-demand loading. This will cancel any uncommitted changes.

c4_ViewRef View**(const char* name_) const**
Used to get or set a named view in this storage object.

void Define(const char* description_)
(Re-) defines the structure of one of the views.

c4_View Get**(const char* name_) const**
Retrieves a named view from this storage object.

void Set**(const char* name_, const c4_View& view_)**
Stores a view under a specified name in this storage object.

void LoadFromStream**(c4_File&)**
Loads contents from the specified input stream.

void SaveToStream**(c4_File&)**
Saves contents to the specified output stream.

c4_Storage::Commit

bool c4_Storage::Commit(void)

Flushes any pending changes to file right now.

Defined in: C:/JC/SRC/MKW/SRC/STORE.CPP

Return Value

Describes commit actions taken:

true
: There were changes, which have been saved to file.

false
: There was nothing to commit, call ignored.

c4_Storage::Contents

c4_RowRef c4_Storage::Contents(void)

Gives access to the stored data as a single row.

Defined in: C:/JC/SRC/MKW/SRC/STORE.CPP

Return Value

Returns a reference to the entire row of the root table.

c4_Storage::Get

d4_inline c4_View c4_Storage::Get(const char* ***name_*****)**

Retrieves a named view from this storage object.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Return Value
Returns a reference to the specified view.

Parameters
*name_*
 the name of the view property to get

c4_Storage::LoadFromStream

void c4_Storage::LoadFromStream(c4_File& ***file_*****)**

Loads contents from the specified input stream.

Defined in: C:/JC/SRC/MKW/SRC/STORE.CPP

Parameters

*file_*
    the stream to read serialized data from

c4_Storage::Rollback

bool c4_Storage::Rollback(void)

(Re)initialize for on-demand loading.

Defined in: C:/JC/SRC/MKW/SRC/STORE.CPP

Return Value

Describes rollback action taken:

true
: Everything was committed, no changes were discarded.

false
: Rolled back one or more uncommitted changes.

Developer Notes

doesn't work in all situations

c4_Storage::SaveToStream

void c4_Storage::SaveToStream(c4_File& *file_***)**

Saves contents to the specified output stream.

Defined in: C:/JC/SRC/MKW/SRC/STORE.CPP

Parameters

*file_*
: the stream to write serialized data to

c4_Storage::Set

d4_inline void c4_Storage::Set(const char* *name_***, const c4_View&** *view_***)**

Stores a view under a specified name in this storage object.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Parameters

*name_*
: the name of the view property to set

*view_*
: the view to be stored

c4_Storage::View

d4_inline c4_ViewRef c4_Storage::View(const char* ***name_*****)**

Used to get or set a named view in this storage object.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Return Value

Returns a reference to the specified view.

Parameters

*name_*
: the name of the view property to use

c4_Streamer Class

class c4_Streamer

An encapsulation of the archive object with additional context.

Defined in: C:/JC/SRC/MKW/SRC/PERSIST.H

c4_StringProp Class

class c4_StringProp: public c4_Property

Class of objects representing string properties.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.H

Class Members
Public members

c4_StringProp**(const char*)**
Constructs a new property.

c4_Row operator[](const char*)
Create a row with a single string.

c4_StringRef operator()(c4_RowRef)
Gets or sets a string property of a row.

c4_StringProp::c4_StringProp

d4_inline c4_StringProp::c4_StringProp(const char* ***name_*****)**

Constructs a new string property.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Parameters

*name_*
    the name of this property

c4_StringRef Class

class c4_StringRef: public c4_Reference

Used to get or set string values.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEWX.H

Class Members
Public members

c4_StringRef(const c4_Reference&)
Constructor.

c4_Table Class

class c4_Table

Tables organize the data and know their own structure and context.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4TABLE.H

Class Members
Public members

Construction / destruction / initialization

c4_Table()
Constructs a new object.

~c4_Table()
Destructor.

void DefineRoot(const char*, c4_Persist* =0)
Initializes this table as a root table.

void DefineSub(c4_Field&, c4_Table&, int)
Initializes this table as a nested table.

void Restructure(c4_Field&)
Restructures a table to match the given field layout.

Table properties

c4_Persist* Persist() const
Returns the associated storage, if any.

c4_Field& Definition() const
Returns the repeating field definition.

c4_Table& Owner() const
Returns a reference to the owner of this table.

int OwnerRow() const
Returns the row index in the owning table.

int OwnerColumn() const
Returns the column index in the owning table.

Table dimensions

int NumRows() const
Returns the number of rows in this table.

int NumColumns() const
Returns the number of columns in this table.

Access to subfields

c4_Field& Field(int) const
Returns the field definition of a subcolumn.

c4_Column& Column(int) const
Returns a specific columns of this table.

bool IsNested(int) const
Returns true if a column consists of subtables.

c4_Table& SubTable(int, int)
Returns the N-th subtable of a subcolumn, creates if necessary.

Table persistence

void Commit(c4_Streamer&, int =0)
Saves current table state to file.

void Prepare(c4_Streamer&)
Loads table state from file.

uchar* LoadColumn(int) const
Loads one column on demand.

Internal methods

c4_HandlerSeq& Sequence()
Returns a reference to the associated sequence.

void ReplaceEntry(int colNum_, int index_, c4_Sequence* seq_)
Replaces the specified subentry with a new sequence.

c4_TableEntry Class

class c4_TableEntry

The table entry is used internally for arrays of subtables

Defined in: C:/JC/SRC/MKW/SRC/PERSIST.H

c4_View Class

class c4_View

A general class of objects to represent a collection of data rows.

Views act like pointers to the actual collections, setting a view to a new value does not alter the collection to which this view pointed previously.

The protocol used for this class mimics the way many other collection classes are defined. For example, when used with MFC, you will see member definitions such as GetSize, GetAt, InsertAt, etc.

The elements of views can be referred to by their 0-based index, which produces a row-reference of type c4_RowRef. These row references can be copied, used to get or set properties, or dereferenced (in which case an object of class c4_Row is returned). Taking the address of a row reference produces a c4_Cursor, which acts very much like a pointer.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.H

Example
The following code creates a view with 1 row and 2 properties:

```
c4_StringProp pName ("Name");
c4_IntProp pAge ("Age");

c4_Row data;
pName (data) = "John Williams";
pAge (data) = 43;

c4_View myView;
myView.Add(row);
```

Class Members
Public members

Construction / destruction / assignment

c4_View**(c4_Sequence* =0)**
Constructs a view based on a sequence.

c4_View**(const c4_Property&)**
Constructs an empty view with one property.

c4_View**(const c4_View&)**
Constructs a view from another one.

~c4_View**()**
Destructor.

c4_View& operator=**(const c4_View&)**
Makes this view the same as another one.

Getting / setting the number of rows

int GetSize**() const**
Returns the number of entries.

int GetUpperBound**() const**
Returns highest index (size - 1).

void SetSize**(int, int =-1)**

Changes the size of this view.

void RemoveAll**()**
Removes all entries (sets size to zero).

Getting / setting individual elements

c4_RowRef GetAt**(int) const**
Returns a reference to specified entry.

c4_RowRef operator[]**(int) const**
Shorthand for GetAt.

void SetAt**(int, c4_RowRef)**
Changes the value of the specified entry.

c4_RowRef ElementAt**(int)**
Element access, for use as RHS or LHS.

These can increase the number of rows

void SetAtGrow**(int, c4_RowRef)**
Sets an entry, growing the view if needed.

int Add**(c4_RowRef)**
Adds a new entry at the end.

Insertion / deletion of rows

void InsertAt**(int, c4_RowRef, int =1)**
Inserts one or more copies of an entry.

void RemoveAt**(int, int =1)**
Removes entries starting at the given index.

void InsertAt**(int, c4_View*)**
Inserts a copy of the contents of another view.

Dealing with the properties of this view

int NumProperties**() const**
Returns the number of properties.

int NthProperty**(int) const**
Returns the id of the N-th property.

int FindProperty**(int)**
Finds the index of a property, given its id.

c4_String Describe() const
Returns a description of the structure.

friend c4_View operator,(const c4_View&, const c4_Property&)
Returns a view like the first, with a property appended to it.

Derived views

c4_View Sort**() const**
Creates view with all rows in natural (property-wise) order.

c4_View SortOn**(const c4_View&) const**
Creates view sorted according to the specified properties.

c4_View SortOnReverse**(const c4_View&, const c4_View&) const**
Creates sorted view, with some properties sorted in reverse.

c4_View Select**(c4_RowRef) const**
Creates a view with rows matching the specified value.

c4_View SelectRange**(c4_RowRef, c4_RowRef) const**
Creates a view with row values within the specified range.

c4_View Project**(const c4_View&) const**
Creates a view with the specified property arrangement.

c4_View ProjectWithout**(const c4_View&) const**
Creates a derived view with some properties omitted.

int GetIndexOf**(c4_RowRef) const**
Returns the index of the specified row in this view (or -1).

Searching

int Find**(c4_RowRef, int start_ =0) const**
Finds index of the the next entry matching the specified key.

int Search**(c4_RowRef) const**
Searches for key, assuming the view is sorted accordingly.

c4_View::Add

int c4_View::Add(c4_RowRef *newElem_*)

Adds a new entry at the end.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Parameters

*newElem_*
    the element to add

c4_View::c4_View

Constructs a new view object.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Syntax

c4_View::c4_View (c4_Sequence* *seq_* =0);

c4_View::c4_View (const c4_Property& *prop_*);

c4_View::c4_View (const c4_View& *view_*);

Parameters

*seq_*
 the sequence

*prop_*
 the property

*view_*
 source for copy constructor

c4_View::ElementAt

d4_inline c4_RowRef c4_View::ElementAt(int *index_*)

Element access, for use as RHS or LHS. Shorthand for GetAt().

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Return Value
A reference to the specified row in the view.

Parameters
*index_*
the zero-based row index

c4_View::Find

int c4_View::Find(c4_RowRef *crit_*, int *start_*)

Finds index of the next entry matching the specified key, using linear search. Only the properties present in the search key are used to determine whether a row matches the key.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Return Value
Returns the index if found, else -1.

Parameters

*crit_*
  the value to look for

*start_*
  the index to start with

c4_View::FindProperty

d4_inline int c4_View::FindProperty(void)

Finds a property, given its id, as returned by NthProperty().

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Return Value

The index of the property, or -1 of it was not found.

c4_View::GetAt

d4_inline c4_RowRef c4_View::GetAt(int *index_*)

Returns a reference to specified entry.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Return Value
A reference to the specified row in the view.

Parameters
*index_*
    the zero-based row index

c4_View::GetIndexOf

int c4_View::GetIndexOf(c4_RowRef& ***row_*****)**

Returns the index of the specified row in this view (or -1). This function can be used to "unmap" an index of a derived view back to the original underlying view.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Return Value
The index of the specified row in this, or -1 of not found.

Parameters
*row_*
  the row to relate to this view

c4_View::GetSize

d4_inline int c4_View::GetSize(void)

Returns the number of entries in this view.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Return Value
A non-negative integer.

c4_View::GetUpperBound

d4_inline int c4_View::GetUpperBound(void)

Returns the highest index in this view (same as: GetSize() - 1).

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Return Value
One less than the number of entries. If there are no entries, the value -1 is returned.

c4_View::InsertAt

d4_inline void c4_View::InsertAt(int *index_*, **c4_RowRef** *newElem_*, **int** *count_***)**

Inserts one or more copies of an entry. This is identical to inserting the specified number of default entries and then setting each of them to the new element value passed as argument.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Parameters

*index_*
    the zero-based row index

*newElem_*
    the value to insert

*count_*
    the nuber of entries to insert (default is 1)

c4_View::NthProperty

d4_inline int c4_View::NthProperty(int *index_*)

Returns the N-th property (using zero-based indexing).

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Return Value
A reference to the specified property.

Parameters
*index_*
    the zero-based property index

c4_View::NumProperties

d4_inline int c4_View::NumProperties(void)

Returns the number of properties present in this view.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Return Value
A non-negative integer.

c4_View::operator=

c4_View& c4_View::operator=(const c4_View& ***view_*****)**

Makes this view the same as another one.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Parameters

*view_*
    the view to use as source

c4_View::operator[]

d4_inline c4_RowRef c4_View::operator[](int *index_*)

Shorthand for GetAt.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Return Value

A reference to the specified row in the view. This reference can be used on either side of the assignment operator.

Parameters

*index_*
    the zero-based row index

c4_View::Project

c4_View c4_View::Project(const c4_View& *in_*)

Creates a view with the specified property arrangement.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Return Value
The derived view, based on the rows of the underlying one.

Parameters
*in_*
    the properties included in the result

c4_View::ProjectWithout

c4_View c4_View::ProjectWithout(const c4_View& ***out_*****)**

Creates a view with the specified property arrangement.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Return Value
The derived view, based on the rows of the underlying one.

Parameters
*out_*
the properties excluded from the result

c4_View::RemoveAll

d4_inline void c4_View::RemoveAll(void)

Removes all entries (sets size to zero).

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

c4_View::RemoveAt

d4_inline void c4_View::RemoveAt(int *index_*, int *count_*)

Removes entries starting at the given index. Entries which have other view references may cause these views to be deleted if their reference counts drop to zero because of this removal.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Parameters

*index_*
the zero-based row index

*count_*
the nuber of entries to remove (default is 1)

c4_View::Search

int c4_View::Search(c4_RowRef *crit_*)

Searches for a key, assuming the view is sorted accordingly. The current version is not very efficient, since it does not take advantage of the fact that the view is sorted (no binary search yet).

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Return Value
Returns index of first row >= crit_, else indexes past the end.

Parameters
*crit_*
the value to look for

c4_View::Select

c4_View c4_View::Select(c4_RowRef *crit_*)

Creates a view with rows matching the specified value.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Return Value
The derived view, based on the rows of the underlying one.

Parameters
*crit_*
    the values determining which row to select

c4_View::SelectRange

c4_View c4_View::SelectRange(c4_RowRef *low_*, c4_RowRef *high_*)

Creates a view with row values within the specified range.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Return Value
The derived view, based on the rows of the underlying one.

Parameters
*low_*
    the values of the lower bounds (inclusive)
*high_*
    the values of the upper bounds (inclusive)

c4_View::SetAt

d4_inline void c4_View::SetAt(int *index_*, c4_RowRef *newElem_*)

Changes the value of the specified entry. If the new value has other properties, these will be added to the underlying view.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Parameters

*index_*
the zero-based row index

*newElem_*
the row to copy to this view

c4_View::SetAtGrow

void c4_View::SetAtGrow(int *index_*, **c4_RowRef** *newElem_***)**

Sets an entry, growing the view if needed.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Parameters

*index_*
  the position of the new entry

*newElem_*
  the element to copy into this view

c4_View::SetSize

d4_inline void c4_View::SetSize(int *newSize_*, int *growBy_*)

Changes the size of this view.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Parameters

*newSize_*
    the new size of this view

*growBy_*
    the granularity of size increases (not used)

c4_View::Sort

c4_View c4_View::Sort(void)

Creates view with all rows in natural (property-wise) order.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Return Value
The derived view, based on the rows of the underlying one.

c4_View::SortOn

c4_View c4_View::SortOn(const c4_View& ***up_*****)**

Creates view sorted according to the specified properties.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Return Value
The derived view, based on the rows of the underlying one.

Parameters
*up_*
    the view which defines the sort order

c4_View::SortOnReverse

c4_View c4_View::SortOnReverse(const c4_View& *up_*, **const c4_View&** *down_***)**

Creates sorted view, with some properties sorted in reverse.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Return Value
The derived view, based on the rows of the underlying one.

Parameters
*up_*
  the view which defines the sort order

*down_*
  subset of *up_*, defines reverse order

c4_View::~c4_View

d4_inline c4_View::~c4_View(void)

Destructor.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

c4_ViewProp Class

class c4_ViewProp: public c4_Property

Class of objects representing view properties.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.H

Class Members
Public members

c4_ViewProp**(const char*)**
Constructs a new property.

c4_Row operator[](const c4_View&)
Create a row with a single view.

c4_ViewRef operator()(c4_RowRef)
Gets or sets a view property of a row.

c4_ViewProp::c4_ViewProp

d4_inline c4_ViewProp::c4_ViewProp(const char* ***name_*****)**

Constructs a new view property.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEW.INL

Parameters

*name_*
  the name of this property

c4_ViewRef Class

class c4_ViewRef: public c4_Reference

Used to get or set view values.

Defined in: C:/JC/SRC/MKW/INCLUDE/K4VIEWX.H

Class Members
Public members

c4_ViewRef(const c4_Reference&)
Constructor.

operator

c4_View operator(void)

Defines a column for a property. This operator is a shorthand for **FindProperty** with the *mayAdd_* parameter set to true.

Defined in: C:/JC/SRC/MKW/SRC/VIEW.CPP

Return Value
This object.

Example
The following code defines an empty view with three properties:

```
c4_IntProp p1, p2, p3;
c4_View myView = (p1, p2, p3);
```

See Also
c4_Property