Concept - “the” Application Server

Edi “Concept” Suica

February 10, 2016

Preface

> *Sometimes I'll start a sentence, and I don't even know where it's going. I just hope I find it along the way - Michael Scott*

About 10 years ago I have started a fun little project - a programming language that breaks the application into user interface and cloud core, enabling you to create on-line applications just like creating any other desktop application. That little project grew into a complete environment for creating, deploying, and managing enterprise applications. I then realized that this powerful technology is impossible to use for everyone else except me, by lack of documentation. I've decided to explain, as simple as possible, how it works and how you can write powerful cloud applications using Concept Programming Language, Concept Framework and all the tools.

You will find real life examples based on commercial solutions implemented by me for different businesses, covering CRMs, ERPs and VoIP applications. I will take you from the basic "hello world" to advanced applications, covering everything, from programming language to framework and advanced mobile applications.

This being said, I hope you are or will become "classy" programmer, who knows that providing a solution isn't enough if it isn't optimal and elegant.

Contents

Part I

Concept Application Server

Chapter 1

Architecture

Concept Application Server (CAS) splits the application into the business part and the interface thin-client. The business part is in fact the application, with unlimited number of interfaces. The business part communicates with the thin client (called Concept Client) by using the Concept data secured protocol. The synchronization is achieved by using a messaging service able to exchange messages both over the cloud (Internet) and over the server itself. No piece of Concept code will be executed on the client, but all the events and user data input will be done with the client.

Concept Application Server has a relatively small resemblance with the traditional HTTP servers, because instead of typical "touch and go" scripts (a script is run for every user event), the application is always active ensuring a better end-user experience. Can be regarded as a bi-directional protocol while HTTP is request-based (unidirectional).

CAS comes with three additional services in order to ensure full efficiency for enterprise applications.

Concept CGI
in order to create Web 2.0 applications

Concept CLI
command line interpreter

Concept Services
support for background tasks that may run without an explicit call from a user

From the developer's point of view, this technology can create complex on-line software with minimum effort. Every single GUI Concept Application runs on-line. Also, it is able to integrate with the typical browser, in order to run applications by simply typing the address.

eg:
concept://devronium.com/HelloWorld/HelloWorld.con

1.1 The server

The Concept Application Server process runs as conceptserver on Unix-derivate operating systems or as concept-server.exe on MS Windows. Starting with Concept 4.0, the server is written in Concept, running as ConceptServer.con both on *nix and Windows. By default it listens on TCP port 2662 (CONC on a phone keypad). It manages the protocol negotiation (secure key-exchange) where a client makes a request, and creates a separate user process (called runsafe).

It can be configured by editing concept.ini located in Program Files/Concept/Bin or /usr/local/etc, depending on the operating system. All the paths in the .ini file are relative to the concept server executable.

You may want to change some of them, for example:

ServerRoot
default server root (eg: ServerRoot=”../Public”)

HostRoot(host)
host root for a specific host (eg: HostRoot(127.0.0.1)=”../Samples”)

Port
the port used by the server (default 2662)

IPv6
set to 1 if you want the server to accept IPv6 connections

Interface
interface to listen on (if not set, will accept connections on any interface)

User
the user under which the server will run

MaxConnection
the maximum number of connections

MaxInitialIDLE
the protocol negotiation timeout

UseSharedMemoryPool
set to 1 if you want for the server to use shared memory pools

The shared pool reduces the memory usage of the server, by using the same space to store the code of different instances of the same application.

The server operates an inter-application message queue, that enables multiple instances of the same application to communicate with each other.

As of version 2.6, the server is capable of restoring dropped connection, for example when the user switches from a network connection to another, without closing the application. This is especially useful in mobile applications, when the users switches from cell network to WiFi.

1.2 Concept protocol

CAS uses a total of 4 protocols:

concept://
default protocol

concepts://
secured protocol

http(s)://
http as a transport layer for concept:// packets (deprecated in Concept Client 3.0)

http(s):// + websocket
websocket used as a transport layer for concept packets (new in Concept Client 4.0)

Everything starts with a protocol negotiation. The server sends a public key(RSA), the client generates a random AES 128-256 key and sends it to the server. The server uses the private key to decrypt the key, and then we have a symmetric secured channel. After the negotiation, the client request the run of a specific application, for example:

run CIDE/cide.con

The server then runs the application and creates an isolated process. The communication is based on the exchange of messages based on four parameters: **Sender**(string), **MessageID**(integer), **Target**(string), **Value**(string). At this point the protocol is entirely binary, and based on message exchange that have the following structure:

Bytes	4	1	o size	4	2	t size	
	message size	o size	owner	message	t size	target	value

Let's say the server wants to get the text in an edit box on the client screen:

```
name = editName.Text;
```

This is what is actually executing (low-level):

```
SendMessage(editName.ID, MSG_GET_PROPERTY, P_TEXT, "");
WaitMessage(editName.ID, MSG_GET_PROPERTY, P_TEXT, name);
```

When using concept secured protocol, the messages are encrypted using the shared AES key.

http protocol is used as an alternative, when running on unstable connections or on networks providing only http connectivity(*figure 1.1*).

Note that a standard concept:// connection can be at any time upgraded to a SSL/TLS connection via the *UpgradeTLS* API discussed later in this book.

Stating with Concept 4.0 (in BETA when writing this book), websockets are now supported. This is useful when using the Concept Client 4.0 JS client, supported in major modern browsers: Firefox, Chrome, Internet Explorer 11, Safari and any webkit-based browser. By default, the Concept Server listens on port 2680, and any application, may be opened directly in browser. If, for example, one would want to run concept://localhost/MyProjects/TestApp/TestApp.con directly in a browser, should simply type http://localhost:2680/MyProjects/TestApp/TestApp.con in the address bar.

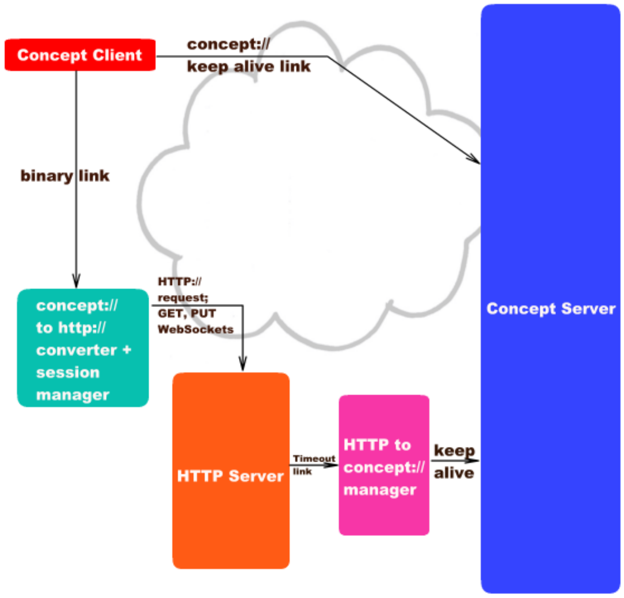

Concept Client
concept://
keep alive link
binary link
concept://
to http://
converter +
session
manager
HTTP://
request;
GET, PUT
WebSockets
HTTP Server
Timeout
link
HTTP to
concept://
manager
keep
alive
Concept Server

Figure 1.1:

1.3 Security

The concept cryptographic system uses RSA keys for initial protocol negotiation, then switches to AES-128. In the 3.0 release you can use AES-192 keys and possibly 256. For RSA 1024 bit keys are used by default, but can be used up to 8192 bits.

Starting version 4.1, Concept supports RSA keys up to 16384 bits and ECC (default) up to 521 bits. In future versions support for RSA keys less than 2048 bits will be dropped.

For today's standards 128 bits is both secure and fast, to avoid CPU overhead. A brute-force attack on the AES-128 key means 3.4×10^{38} possible combinations, prohibitive for this type of attack for today's computing power.

A subsystem is used for DRM (Digital Rights Management). Concept is able to selectively encrypt packages. For example, in a VoIP application, you may opt to encrypt only the voice packages, reducing the CPU overhead. This is useful in mobile application, where hand-held devices may use a little more battery for encryption.

The user login APIs uses digest MD5 and SHA1 algorithms, both in plain form or challenge response authentication. This can be selected by the programmer by selecting an appropriate login method.

1.4 Core

The core is a virtual machine that uses both an interpreter and a JIT compiler. The VM operates on byte code generated by the Concept Compiler (called accel). It has a structure composed of one operator, two working operands (called left and right), a reserved operand and a result. The keywords are treated as special operators without return, in order to have a fixed structure, optimizing the execution time. When the interpreter executes a function for the second time, tries to compile the code in order to maximize speed and memory usage. Functions that are called just one time are not compiled to native code in order to save memory and CPU power. The control can be transferred in and out between interpreter and native code without causing problems. The core does not use a fixed memory pool, the application being free to allocate any amount of memory. In order to save memory it uses shared memory pools, loading the byte code only once regardless of the number of users connected and running different instances of the same application. All the variables created by executing code are accounted for, and the application cannot access the actual address of a variable. In practice, every variable is a pointer, but the programmer deals only with the immediate value.

1.5 Concept Assembly

The byte code generated by the Concept Compiler is called Concept Assembly. Is a mix of operators and instructions like IF and GOTO. For example:

```
i=a+b+c*d;
if ((i>10) || (i<-10))
   echo "Hello!";
```

will be optimized and decomposed to:

```
t0=c*d
t1=a+b
i=t0+t1
IF (I>10, ECHO..., IF(I<-10, ECHO...))
```

(Note that if first expression is TRUE the second one will not be evaluated)

The resulting Concept Assembly:

Instruction	Operator	Left	Right	Result	Jump
1	*	c	d	t0	
2	+	a	b	t1	
3	+	t0	t1	i	
4	>	i	10	t3	
5	IF	t3			7 (if false)
6	GOTO				10
7	<	i	-10	t4	
8	OR	t3	t4	t5	
9	IF	t5			11 (if false)
10	ECHO	”Hello!”			
11	RETURN				

1.6 Garbage collector and smart data linking

Concept has two memory management strategies: a kind of smart pointer (that increase and decrease reference count) called SDL and a garbage collector for managing cyclic references. Immediate cyclic references(object that reference itself) are managed with ease by SDL. However cyclic references that span over two ore more objects are more difficult to manage. Concept variables have some overhead. Internally, the same structure is used to keep any variable. For each of these variables, a link count, a type flag, and property data overhead are kept. This means 15 bytes of metadata info (on 64 bit platforms). When a variable link count is zero, the memory is automatically freed. However there is a case that could generate leaks - the cyclic references. These are freed by the garbage collector.

An example of cyclic reference:

```
arr=new [];
someobject.references=arr;
arr[0]=someobject;
```

This situation will be managed by the Garbage Collector at some point of program execution. In my opinion, this is bad programming, but is a situation we all encountered sooner or later.

At various intervals, the garbage collector will check the variable reachability and if some variables are unreachable, it will automatically free the them. The garbage collector can also be manually invoked via CheckReachability function:

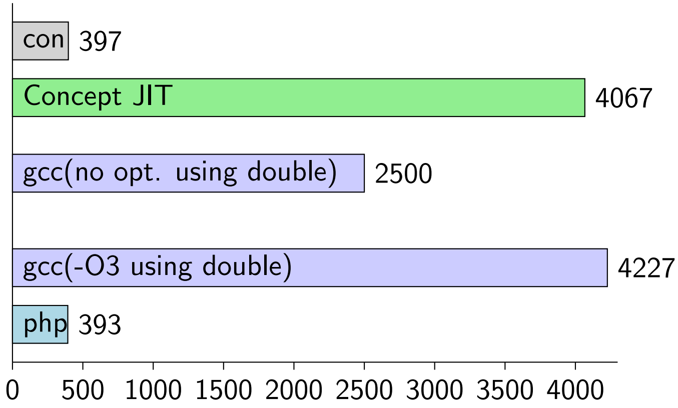
```
CheckReachability();
```

Be advised that is not recommended to call this function, except for memory critical complex data structures applications. Also, on large datasets, this function call will be slow,

1.7 JIT compiler

In 2013 I've realized that the interpreter has reached a point where it couldn't go faster. Then I've decided to use a Just-In-Time compiler to generate code specific to the platform(currently tested on i386, amd64 and ARM). After implementing an open source JIT(called sljit), the performance raised noticeably. I've used a sieve benchmark to measure the gain in performance, and it was about 10 times faster than before, with scores comparable to native code(binary code generated by gcc/g++). The main problem was that Concept stored all its numeric variables as double precision floating points. In order reduce the number of executed instructions, the system tries to combine two or more instructions. For example, the JIT compiler identifies a vector initialization in a loop, and instead of iterating in that loop, it calls a special function that automatically initializes the array.

The results for the sieve benchmark implementation(using doubles).



1.8 Modules

The Concept core can be easily extended by writing modules in virtually any programming language that has a native compiler. The standard server has about 80 modules that cover specific operations ranging from basic I/O to high level modules like Twitter interaction.

It is fairly easy to create a Concept module. For example, in C/C++ you can create two files:

library.h

```
#ifndef __LIBRARY_H
#define __LIBRARY_H

// provided with concept, defines all
// the macros and data structures
#include "stdlibrary.h"

extern "C" {

   CONCEPT_FUNCTION(sin)
   CONCEPT_FUNCTION(cos)
}
#endif
```

main.cpp

```
#include "library.h"
#include <math.h>

CONCEPT_FUNCTION_IMPL(sin, 1)
    T_NUMBER(0)
    RETURN_NUMBER(sin(PARAM(0)))
END_IMPL

CONCEPT_FUNCTION_IMPL(cos, 1)
    T_NUMBER(0)
    double result=cos(PARAM(0))
    RETURN_NUMBER(result)
END_IMPL
```

Then, you just need to compile it:
On windows:
gcc -shared main.cpp -o eduard.library.test.dll

Everywhere else:
gcc -shared main.cpp -o eduard.library.test.so

Creation of modules will be discussed in detail in **Chapter 6 - Static functions**.

The list of standard modules:

Module name	Depends	Opt.	Notes
concept.helper.idgenerator			Generates object IDs
standard.ai.fann	libfann	YES	Artificial neural networks
standard.arch.opus	libopus		OPUS audio codec
standard.arch.speex	libspeex		SPEEX audio codec
standard.arch.h264	OpenH264	YES	H.264 video codec
standard.C.casts			cast functions (C/C++)
standard.C.io			basic I/O functions
standard.C.math			math functions
stabdard.C.string			C string functions
standard.C.time			time functions (C/C++)
standard.coding.base64			mime-encode
standard.db.dbase		YES	dBase support
standard.db.firebird		YES	Firebird driver

standard.db.mongo		YES	Mongo driver
standard.db.mysql		YES	MySQL driver
standard.db.nuo		YES	NuoDB driver
standard.db.pq		YES	PostgreSQL driver
standard.db.sql			ODBC driver
standard.db.sqlite		YES	SQLite driver
standard.graph.imagemagick	MagickWand	YES	ImageMagick interface
standard.graph.svg	librsvg	YES	SVG support
standard.graph.svgt	libsvgt	YES	Tiny SVG support
standard.graph.wk	webkit	YES	Server-side web page screenshot
standard.io.rs232		YES	rs232 interface
standard.lang.cli			Command-line utils
standard.lang.js	spidermonkey	YES	JavaScript functions
standard.lang.parallel	OpenCL	YES	Parallelization functions
standard.lang.profiler		YES	Profiler support
standard.lang.serialize	libxml2		Serialization utils
standard.lib.amf		YES	AMF serialization
standard.lib.captcha			Captcha generator
standard.lib.chart	libgd	YES	Deprecated chart engine
standard.lib.cripto			Criptographic library
standard.lib.csv			CSV support
standard.lib.dtmf			DTMF supports (telephony)
standard.lib.face	OpenCV	YES	Face detection functions
standard.lib.gd	libgd	YES	Deprecated
standard.lib.hpdf	libharu	YES	Deprecated PDF generator
standard.lib.hunspell	hunspell		Spell checker
standard.lib.iconv	libiconv		Text conversion
standard.lib.json	json-c		JSON serialization
standard.lib.languagedetector	CLD		Language detector
standard.lib.msword		YES	MS Word document parsing
standard.lib.ocr	tesseract	YES	OCR
standard.lib.poppler	libpoppler		PDF
standard.lib.pcre	libpcre		Perl regexp
standard.lib.regex			POSIX regexp
standard.lib.shared			Shared memory functions
standard.lib.sphinx	PocketSPHINX	YES	CMU Sphinx voice recognition
standard.lib.str			String manipulation
standard.lib.thread			Multi-threading
standard.lib.virt	libvirt	YES	Virtualization interaction
standard.lib.xls	libxls	YES	Excel file support

standard.lib.xml	libxml2		XML
standard.lib.xslfo		YES	XSL:Fo
standard.lib.xslt	Sablotron	YES	XSLT
standard.lib.xslt2	libxslt		XSLT
standard.math.gmp	libgmp	YES	
standard.math.rand			Random numbers support
standard.net.bluetooth	bluez	YES	Bluetooth support
standard.net.curl	libcurl		Multiple protocols
standard.net.dns		YES	DNS protocol
standard.net.ftp		YES	FTP protocol
standard.net.geoip	GeoIP	YES	IP tracking
standard.net.im	libpurple	YES	IM protocols
standard.net.ldap		YES	LDAP support
standard.net.mail		YES	POP3/SMTP protocol
standard.net.mapi		YES	
standard.net.memcached	libmemcached		Memcached client
standard.net.modbus	libmodbus	YES	Modbus protocol
standard.net.flow		YES	Netflow v5/v9 parser
standard.net.opal	OpalSIP	YES	SIP and RTP protocols
standard.net.rtp	jrtplib	YES	RTP protocol
standard.net.sip	libosip2	YES	SIP protocol
standard.net.snmp	netsnmp	YES	SNMP protocol
standard.net.soap	C-SOAP	YES	SOAP
standard.net.socket			Socket support
standard.net.ssh	libssh2	YES	SSHv2
standard.net.twitter	libcurl	YES	Twitter protocol
standard.net.webdav		YES	WebDav support
standard.search.xapian	Xapian		Xapian search
standard.tts.mbrola	mbrola	YES	Text to speech engine
web.server.api			Web applications helper
web.service.template			Deprecated smarty-like templates
win32.base.message			Dispatching client messages
win32.graph.freeimage	FreeImage		Image manipulation functions

Note that modules prefixed by ”win32” have nothing to do with Windows. They are named this way purely for backward compatibly only.

1.9 Supported platforms

Concept Application Server runs on various operating systems and CPU architectures. For example, I like how it performs on FreeBSD, but the majority of the actual users runs Concept of Linux (Debian, Ubuntu) and Windows for its simplicity. However, a while ago I've moved a CAS solution from Windows 2008 physical server to a Linux virtual machine with similar configuration. I was more that impressed on how it performed. It seemed to be at least twice as fast when dealing with databases. It seems that Linux does a better job when caching the data than Windows. I'm not a open-source "politick" freak, I actually consider that most of the operating systems are good at what they do: interface the user with the hardware.

You will find on *devronium.com* website binary packages for Windows, Linux, in both 32 and 64 bit flavors. For everything else, you will have to download and compile the sources. I've compiled the sources with little to no modifications on FreeBSD(intel), OS X(intel) and Linux(ARM). In theory it should work on PPC and SPARC, but I had no request yet for that platforms. It supports both big endian and little endian platforms.

On Microsoft Windows, you just have to download the Concept Server installer from devronium.com and open it. On Debian and Ubuntu, you may download the appropriate binary package (i386 or amd64) and install it using:

```
$ sudo dpkg -i concept.server.3.0.amd64.deb
```

And then, to resolve dependencies:

```
$ sudo apt-get install -f
```

As a note, the only framework element not cross-platform is the UNIX socket (discussed further in this book). On Windows, UNIX sockets are emulated using named pipes, but have limited support when compared to real unix sockets.

Chapter 2

The client

Concept client “the bee” (see figure 2.1) is a thin client that allows the user to access concept:// applications.

Concept client is cross-platform and has similar behavior on Windows, OS X, Linux/Unix, BSD, Android and iOS.

There are two different approaches in how the client is implemented. First, there is the desktop version, that allows plug-ins and feature-rich controls. Then, there is the mobile version, that is focused on saving battery power and provide a consistent user experience by using only native controls.

Note: Concept client uses cryptography and may be illegal in some countries.

Figure 2.1:
Concept Client Bee

Starting with Concept Server 4.0 a new client was introduced. Concept Client 4.0 JS is a pure JavaScript implementation of the concept protocol, on top of web sockets. Almost any native concept:// application could be run directly in browser.

2.1 Workstations

Concept Client can be run on most desktop operating systems, including Microsoft Windows, Mac OS X and Linux. Its user interface is based on GTK+ for versions up to 2.6 and on Qt for 3.0. Both GTK+ and QT provide a cross-platform framework that can use native controls (on Windows and Mac OS X).

The Bee can use both direct address or saved shortcuts. For example, if you click on "Open concept address" on Windows or ConClient on Mac OS X, a prompt will ask you to enter an addres. An example address would be *concept://devronium.com/HelloWorld/HelloWorld.con*. Alternatively you can use server shortcuts - text files with the .ss extension. **HelloWorld.ss** opened as a text file.

```
[Shortcut]
Host        =  devronium.com
Application =  HelloWorld/HelloWorld.con
Secured     =  No
```

This is equivalent of using *concept://devronium.com/HelloWorld/HelloWorld.con*. If you want to use the secured version of the protocol(concepts://), you can set the *Secured flag* to *Yes*.

Also, you can pass parameters to an application by adding at the path's end: *concept://devronium.com/HelloWorld/HelloWorld.con?name=Edi*.

Alternatively you can invoke an application from command line, for example:

```
$ conclient concept://devronium.com/HelloWorld/HelloWorld.con
```

or

```
$ conclient devronium.com HelloWorld/HelloWorld.con
```

On Windows the installer associates concept: protocol with the bee, and you can send concept links by e-mail and open it from browsers.

The workstation version has a plug-in system that enables you to add custom controls to a specific client. For example, on Windows version you can use COM (.NET) objects, but I wouldn't recommend it because of the portability issue. By default, Concept Client comes with the following plug-ins:

Audiere
audio playback (built-in in Concept Client 3.0)

Audiostream
audio recording and playback

Flash
based on SWFDec (deprecated since 3.0)

Glade
used by the CIDE for design view (replaced by RDesigner in Concept 3.0)

HTML
html rendering component (built-in in Concept Client 3.0)

Julius
speech recognition

Multimedia
audio and video playback based on the FFmpeg libraries (replaced by built-in APIs in Concept Client 3.0)

PDF
pdf rendering component (built-in in Concept Client 3.0)

Scintilla
syntax highlight and auto completion control

Videocapture
video capture and processing based on the OpenCV library

WebKit
browser component (built-in in Concept 3.0)

2.2 Mobile devices

The mobile Bee is just a stripped down version of the Concept Client that drops GTK+ or Qt, and uses instead the native UI of the target operating system. At the time I wrote this book, it had three versions: one for Android 2.3.6, one for Android 4.x and one for iOS 7 or bigger. A Windows phone version is planned, but I can't confirm yet that will be in production. All these are using the same core as the workstation version (written in C/C++) but has specific user interface code (written in Java for Android and Objective C for iOS).

For security reasons, the mobile version is not expandable (no plug-ins are available), but the following plug-in functionality is built in: audio streaming, WebKit engine, camera and microphone capture. It has some additional features not available on workstation, like user tracking (GPS or Network) and device wake/sleep commands.

It uses the same relative engine used by the workstation version in order to provide consistent layout on different screen resolutions. However, I recommend that the developer create different interfaces for desktop and mobile keeping in mind that screen size is significantly smaller in smart phones and tablets.

2.3 Web browsers

Additionally Concept Client 4.0 JS can run in most modern browser (Firefox, Chrome, IE 11, Safari and any other webkit-based browser). Concept Client 4.0 JS is included in the Concept Server 4.0 standard distribution. Is build in pure JavaScript, using bootstrap as the UI library. It uses web sockets for exchanging data (instead of GET/POST), enhancing the response time. The client is relatively light, having only bootstrap as a dependency, and a few optional plug-ins depending on jQuery. This is soon to be the primary concept client. The only restriction, for now, comes from Safari and Internet Explorer, not supporting audio and/or video APIs, making impossible a full implementation of the concept native client. For most non-voip/video applications this should not be a problem.

All concept development tools (CIDE, GyroGears) run on the JS client, but I still recommend using the native client for development, because the UI is not optimized (yet) for web browsers. Full web browser support is planned for Q3 2015.

Chapter 3

Developer tools

The standard Concept Server distributions comes with tools like Concept IDE, GyroGears, BeeKeeper, DoInstall subsystem and WKB(a simple web browser based on WebKit).

3.1 Concept IDE

CIDE is the Concept Integrated Development Environment. It includes features like:

- syntax highlighting
- code completion
- members/class folding
- design view with Concept RDesigner

The code editor is based on the scintilla control. It communicates with the Concept Debugger, being able to run in debug mode concept server application, console applications and web applications. CIDE is open source and cross platform(like any other Concept application). Due to the Concept Application Server (CAS) working model, a single CIDE distribution may run on a server and be instantiated by multiple software developers.

In addition, it includes some various tools like SQL Tool, a tool that works with most of the database servers (using ODBC). Another useful tool is the Team Chat panel, which allows developers to chat and share code.

CIDE comes by default with Concept Application Server by selecting developing tools when installing. On Windows you can open it locally by running Start -¿ Programs -¿ Concept II -¿ CIDE.

3.2 Application deployment

The deployment is a simple process based on a few steps. If you user GyroGears, the package installer will be created automatically by pressing the ”Create installer” button. On CIDE projects, you must compress the application folder into a zip file. You may create a .ss(server shortcut) file and put it in the archive. Rename the .zip file to .conceptinstall and you have the installer. Upload the resulting file to a web server of your choice.

On the target machine, you can either run the DoInstall.ss shortcut, or send a link by e-mail like this:

```
<a href =
    "concept://localhost/Do/Install?http://yourserver.com/YourApp.conceptinstall">
    Click here to install/update the application
</a>
```

If you manually run DoInstall on the target machine, you can simply paste your link:

```
http://yourserver.com/YourApp.conceptinstall
```

If you bundled a .ss file, then it will be copied into the ”Installed applications” folder. Your application will be unpacked into the Concept server root.

This allows you to fully deploy your application with minimum effort, without needing remote access because the end-user is the one who requests the update. This avoids a lot of firewall problems because the update is made via a simple HTTP request.

Part II

Concept Programming

Chapter 4

Language

The Concept programming language is the heart of the entire system. It is strictly object oriented, with a traditional syntax similar to ECMAScript/Java/C#/C++, being designed to be easy and fun.

Before we start, remember:

- Every application must have exactly one class named Main with an implemented constructor.
- Variable, member and class names are case sensitive. This means that foo is not the same with Foo.
- A class name must be unique for the entire application. You cannot redefine a class, you can just extend it (with a new unique name)
- Member names must be unique in the same class, regardless of type or parameters
- Variable names must be unique function wide regardless where they are declared
- For source files is recommended to use the *.con* extension. For sources used in web pages *.csp* is recommended (acronym from Concept Server Page).

4.1 Hello world

Let’s create our first console application that will write a message to the standard output.
HelloWorld.con

```
class Main {
    function Main() {
        echo "Hello world!";
    }
}
```

The program execution starts by creating one instance of the class Main (somehow a singleton). This results in a call to the Main constructor, Main(). This is your application entry point. Note that Main() should have no parameters.

echo is a keyword. The reason is a keyword and not a module function is that output is managed explicitly by the Concept core, and as you will see in the next example, sometimes the output travels over a socket or a web server.

On non-Windows platform, you could add a special first line enabling you to threat a Concept Program as a shell script. Note that the ”function” keyword is purely optional, so you could simplify this example to:

```
#!/usr/local/bin/concept

class Main {
    Main() {
        echo "Hello world!";
    }
}
```

You can execute this code by typing in a shell.

Windows:

```
HelloWorld.con
```

Non-windows:

```
./HelloWorld.con
```

Note that on non-windows you must execute a chmod first:

```
$ chmod 0755 HelloWorld.con
```

or, explicitly call the core (without setting any execution rights)

```
concept HelloWorld.con
```

This will cause the compilation in memory and then the execute the program.

When dealing with large applications (tens of thousands lines of code) this will become slow. It's recommended that you compile the source code using the concept compiler - **accel**

```
accel HelloWorld.con
```

This will produce a file named *HelloWorld.con.accel*. This file contains the byte code(concept assembly) for the entire program. Note, that you must supply both files (the .con and the.con.accel). If you're working at a closed-source project, just place an empty file named *HelloWorld.con* instead of actual source.

Beginning with Concept 4.1, semicolon is optional if *pragma strict off* is set. The above example could be rewritten as:

```
#!/usr/local/bin/concept
// note the following line
pragma strict off

class Main {
    Main() {
        // note the missing semicolon
        echo "Hello world!"
    }
}
```

For readability, it is preferable to use semicolon, but I noticed there is a trend for dropping it, so it's up to the developer's coding style.

4.2 Hello world from between the clouds

We've seen in the previous section a classic program. Now we will create a similar application that makes use of the concept:// protocol. Let's consider the following example.

HelloWorldCloud.con

```
include Application.con
include RForm.con
include RLabel.con

class HelloForm extends RForm {
    private var labelHello;

    HelloForm(Parent) {
        super(Parent);
        labelHello=new RLabel(this);
        labelHello.Caption="Hello world!";
        labelHello.Show();
    }
}

class Main {
    Main() {
        try {
            var Application=new CApplication(new HelloForm(null));
            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            echo "Didn't catch $Exception";
        }
    }
}
```

This must be saved into your server's root. For localhost connections, the root is "Samples", for remote is "Public" on non-windows. On Windows, it

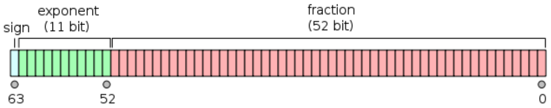


Figure 4.1:
Double floating point representation

is “Samples” for any connection.

You should be able to open (via Concept Client, desktop and mobile, refer to section “The Client”) the link:
concept://localhost/HelloWorldClound.con
Just keep in mind that no code executes on the client. Everything is done on the server, the client just showing an interface to the user.

Don’t be alarmed number of lines. Everything will be written for you by Concept IDE as you will see in the next chapters.

4.3 Data types

Concept data types are simple and easy to switch between them. The types are: number, string, object, array and delegate.

All the numeric values (integer, both signed and unsigned and real numbers) are kept as double precision floating point numbers, known as **double** (IEEE 754 binary64). This should be big enough to handle any kind of number (see figure 4.1). The only disadvantage of this approach is the lower speed in computation based on integers stored as double floating point. However on modern CPUs the floating point operations are now comparable with integer operations.

Maximum integer that can be stored without loosing precision is 2^{53}. Loss of precision will occur after that value. CPU architecture doesn’t affect the number domain.

Strings, objects and arrays use additional overhead bits. To optimize memory usage, each member variable is created when first used.

Strings are limited to a length of about 2,147,483,647 (2^{31}-1) on 32 bit architecture, which is both the maximum value for the signed integer representing its size and maximum memory segment size. On 64 bits the theoretical limit is 2^{63} but in practice depends of operating system and CPU.

Each class has a maximum member count of 65,535 members(regardless of architecture).

Arrays have a maximum element count of 2,147,483,647 regardless of architecture. Note that on 64 bits an array could have more elements, but it won't be managed by the JIT (but could be interpreted). Array string keys must not contain the null character.

A variable is defined with the keyword **var**(short for variant). When declaring a variable, you can set the default value for it, for example:

```
// declaring a numeric variable
var a=10;
// declaring a string variable
var s="Hello !";
// declaring an array
// exactly the same as var arr=new [];
var[] arr;
// or
var arr2=[1, 2, 3, a, s];
// or, with keys
var arr3=["one" => 1, "two" => 2, "three" => 3, "a" => a, "s" => s];
// declaring an empty variable
// exactly the same as var x=0;
var x;
// declaring multiple variables
var i, j=2, k;
```

4.3.1 Numbers

Numeric data type is used to store both real and integer values. All Concept variables with no other value assigned are by default numeric with a value of zero.

SolveSimpleEquation.con

```
class Main {
    Main() {
        var a, b, x;
        a=3;
        b=2;
        // you can say: var a=3, b=2, x;
        // it has the same result of the above code
        // you should test for a not zero
        x=b/a;

        echo "The solution of the equation ${a}x+$b=0 is $x\n";
    }
}
```

Resulting in

```
The solution of the equation 3x+2=0 is 0.666666666666667
```

You can use hexadecimal values by adding the ”0x” prefix.

```
echo 0xFFFF;
```

Will output 65535. Hexadecimal numbers are not case sensitive, 0xFFFF being exactly the same as 0xffff. You cannot represent non-integer numbers as hexadecimal.

Also, numeric values are used as boolean. There are three special constants: **true**(value 1), **false**(value 0), **null**(value 0);

BooleanExample.con

```
class Main {
    Main() {
        var a=true;
        if (a)
            echo "Is true!";
        else
            echo "Is false!";
    }
}
```

4.3.2 Strings

Concept strings are auto-allocable, self managed and easy to use. The string contents must be enclosed either by ' or by " quotes. Simple quotes do not parse the string.

For example **StringExample.con**

```
class Main {
    Main() {
        var name="Eduard";
        echo "Hello $name!";
    }
}
```

will print out

```
Hello Eduard!
```

The same example, using simple quotes

```
class Main {
    Main() {
        var name="Eduard";
        echo 'Hello $name!';
    }
}
```

The output:

```
Hello $name!
```

The $ sign will break the string enclosed by "(first example), and insert there the given value. You can only insert one variable. If you need to make some computation, or use class member variables, you need to user the { }.

StringExampleWithExpression.con

```
class Main {
    Main() {
        var a=1;
        var b=2;
        echo "The result of $a + $b is ${a+b}";
    }
}
```

will print out

```
The result of 1 + 2 is 3
```

You can access characters of the string by using the index operator.

```
class Main {
    Main() {
        var word="Bicycle";
        echo "$word starts with ${word[0]} and ends with
            ${word[length word - 1]}";
    }
}
```

The output:

```
Bicycle starts with B and ends with e
```

You can perform positional updates using the index operator:

```
class Main {
    Main() {
        var word="2 words";
        word[0]="3";
        echo word+"\n";
        word[0]="four";
        echo word+"\n";
    }
}
```

The output:

```
3 words
four words
```

As you noticed, the a character may be replaced by more than one characters.

You can escape a string by using \. For example, *"My name is \$name"* will print out **My name is $name** instead of using the variable name. You can insert specific characters: *"2 to n-th power: 2\xFC"* will output **2 to n-th power:** 2^n.

Character	Notes
\n	New line
\r	Carriage return
\t	Tab character
\x	Hexadecimal character following (next two characters)
\0	Octal character following
\v	Vertical tab
\f	Form feed
\a	Alarm (a beep)
\b	Backspace
\$	Don't evaluate next $ character
\¨	Double quotes escaped
\´	Simple quote escaped

You can find out the length of a string by using the *length* operator (*var len = length str*).

4.3.3 Arrays

Concept vectors are not homogeneous. This means that you can put virtually anything in an array. The same array can contain numeric elements, strings, objects and even references to itself.

There are 3 way to create an array:

```
var first_array=new [];
var[] second_array;
var third_array=[ ];
```

For the given code, all three arrays produce the same result - an empty array.

ArrayExample.con

```
class Main {
   Main() {
      var arr=new [];
      arr[0]=1;
      arr[1]=[1, 2, 3];
      arr[2]="Hello!";
      arr[3]=this;
      echo arr;
   }
}
```

Will output:

```
Array {
        [0] => 1
        [1] =>
                Array {
                        [0] => 1
                        [1] => 2
                        [2] => 3
                }
        [2] => Hello!
        [3] => Main
}
```

The array contains everything, from numbers to another array and a reference to object.

A matrix can easily be defined like this:

```
var matrix= [
    [10, 11, 12, 13],
    [20, 21, 22, 23],
    [30, 31, 32, 33]
];
```

If, for example, the element on the line two, column one is needed, matrix[1][0] may be read or set.

```
echo matrix[1][0];
```

Will print 20 on the screen. Remember that the index of the first element in an array is 0, so matrix[1] will reference the second array (line). All array indexes must be greater than 0. A negative index will result in a run-time error.

Arrays can also have string keys. Note if a key contains a null character, it will be used as a key only until that point.

For example:

```
var a=arr["key\0 1"];
var b=arr["key\0 2"];
```

Will reference the same element, *arr[”key”].*

KeyExample.con

```
class Main {
    Main() {
        var arr=["One": 1, "Two": 2, "Three": 3];
        arr["Four"]=4;
        echo arr;
    }
}
```

Will output:

```
Array {
       [0,"One"] => 1
       [1,"Two"] => 2
       [2,"Three"] => 3
       [3,"Four"] => 4
}
```

Note that you can use **:** or **=>** in key/value pairs,
[”One”: 1, ”Two”: 2, ”Three”: 3]
being exactly the same as
[”One” => 1, ”Two” => 2, ”Three” => 3].

You can find out the length of an array by using the *length* operator (*var len = length arr*). An array element may be accessed by its index or by its key, for example, in the previous example, we could say:

```
class Main {
   Main() {
      var arr=["One": 1, "Two": 2, "Three": 3];
      // access array element by index
      // 0 is the index of the first element
      if (arr[0]==1)
         echo "First element is 1";
      // access array element by key
      if (arr["Two"]==2)
         echo "Element with key 'Two' is 2";
   }
}
```

By using the *KeySorted*(array) function, an new array will be created, with elements ordered by key. This is a very fast way of sorting data, because the key are already sorted in the array index, resulting in a *O(1)* running time.

For example:

```
include Serializable.con
[..]
var arr = KeySorted(["Z": "z", "M": "m", "K": "k"]);
[..]
```

will return:

```
["K": "k", "M": "m", "Z": "z"]
```

The *GetKeys*(array)(must include Serializable.con) static function will return an array containing the keys used in the given array. For a complete list of the array functions, please check the Concept Documentation.

4.3.4 Objects

Object are instances of classes. You can create one by using the **new** operator. A class can have member variables, functions, properties, constructor, destructor and virtual members. Can extend an existing class. Member variables can be of any type. In a single source file you can define any number of classes you need. However, I would recommend that each class should have its own source file.

SimpleObjectExample.con

```
class Repair {
    var Cost=0;
    var Description="";
}

class Car {
    var Maker="";
    var Model="";
    var Driver="Yellow";

    var[] Repairs;
}

class Main {
    var[] cars;
    Main() {
        var car=new Car();
        car.Maker="Mini";
        car.Model="Cooper";
        car.Driver="Mr. Bean";

        var repair=new Repair();
        repair.Cost=30.20;
        repair.Description="Changed the door locks";

        car.Repairs[0]=repair;

        this.cars[0]=car;
    }
}
```

You can access a class member by using the selector operator(point). **this** always refers the current object. This is a special variable created in each function. You cannot define a variable named ”this”.

Note that instead of **.** *you could use* **->** *(C/C++-style).*

Class member and variable names obey the same naming rules. All keywords are reserved and cannot be used as variable names. Variables names must contain only characters a to z, A to z, 0 to 9, and _. Any other character is invalid. Both member and variable names cannot begin with a number. There is no explicit rule on maximum name length, but I would recommend to avoid extremely long names.

4.3.5 Delegates

Delegates are references to existing functions. Both a function and object reference can be stored in a variable.

DelegateExample.con

```
class Main {
    foo() {
        echo "Delegate call!";
    }

    Main() {
        var d;
        d=this.foo;
        d();
    }
}
```

Will output:

```
Delegate call!
```

A delegate variable increments the object links count in order to ensure that the delegate will be called. The link count will be decreased when the delegate variable will be deleted or reset.

Delegates are used by Concept Framework in mapping events. It is a convenient way of referencing both an object and a function defined in object's class.

Keep in mind that a delegate references both the object and the class member and as a consequence it cannot reference a static function.

4.4 Unary operators

Unary operators have only one operand.

Operator	**Data types**	**Can overload**	**Description**
!	all		Logical negation (NOT)
˜	number	YES	Bitwise negation
++	number, string	YES	Increment
–	number	YES	Decrement
@	number		Special prefix
length	string, array		Get length
value	number, string		Evaluate string
typeof	all		Get variable type
classof	class		Get object class
new	class, array		Creates an object
delete	all		Equivalent of object=null

The ! operator applies to all data types. !number returns 1(true) if number is 0(null) or 1(false) otherwise. !string returns true if string is empty, false otherwise. !array returns true if array has no elements, false otherwise. !object always returns false. The operator performs a bitwise negation on an integer. For example, 12 (1100 in binary) the result will be 3(0011 in binary).

! and operators are used always on the left side of the operand

++ and – operators applies to strings and numbers. It can be used both at left or at right side of the operand.

For example:

```
var a=1;
echo ++a;
```

Will increment a to 2, and will print it (2).

When used at right side:

```
var a=1;
echo a++;
```

First will return the current value of a (1) and then will increment it, printing 1.

Similar with –, but instead of incrementing will decrement a by 1.

```
var a=1;
echo --a;
```

Will print 0.

@ prefix (not an operator) will give you access to some compile-time information. You can use it as @line, @filename, @path, @time, @class and @member.

SpecialConstant.con

```
class Main {
   Main() {
      echo @class+"."+@member+":"+@line+", file "+@filename+",
          compiled "+@time;
   }
}
```

Outputs

```
Main.Main:3, file SpecialConstant.con, compiled 1388601305
```

@line will return the line number, @class the class name, @member the member name, @filename the filename, @path the full path and @time the compilation time as seconds since epoch (January 1, 1970, 00:00:00).

length operator is useful when you want to get the length of an array or string.

```
echo length "12345";
```

Will output 5. For arrays:

```
echo length [1, 2, 3, 4, 5];
```

Will output 5.

You cannot use length with objects and delegates.

value operator evaluates a string to a number. When used on a number it simply returns the number.

```
var s="10";
var result=s+1;
```

Will make *result* a string containing ”101”, because it will concatenated 1 to the existing string. But, if you use:

```
var s="10";
var result=value s + 1;
```

Will make *result* a number with the value 11.

typeof operator will return a string describing the type of a given variable. It can be only one of the following values: “string”, “numeric”, “array”, “class” and “delegate”.

typeofExample.con

```
class Main {
   foo() {
   }

   Main() {
      echo typeof "Hello!";
      echo "\n";
      echo typeof 1;
```

```
        echo "\n";
        echo typeof this;
        echo "\n";
        echo typeof this.foo;
        echo "\n";
        echo typeof [1, 2, 3];
    }
}
```

outputs

```
string
numeric
class
delegate
array
```

classof is similar with typeof, but instead of the type it returns the class name for a given object. If the given object is not a class, it will return an empty string.

classofExample.con

```
class Main {
    Main() {
        echo classof this;
    }
}
```

outputs

```
Main
```

The **new** operator is used to create objects or array. On the right side it should have the name of the class to instantiate or [] if you want to create an array. Note that [] must contain no characters or spaces.

newExample.con

```
class Test {
    Test(name) {
        echo "Hello $name!";
    }
}

class Main {
    Main() {
        var arr=new [];
        var obj=new Test("Eduard");
    }
}
```

This creates an empty array(*arr*) and an object(*obj*) and calls Test's constructor, outputting:

```
Hello Eduard!
```

delete exists only for syntax compatibility reasons.

deleteExample.con

```
class Test {
    finalize() {
        echo "Destroyed";
    }
}

class Main {
    Main() {
        var obj=new Test();
        echo "About to delete ... ";
        delete obj;
        echo "done\n";
    }
}
```

Outputs:

```
About to delete ... done
Destroyed
```

As you can see, the Test destructor is called long after the call to delete.

This is because the core manages the memory. Setting obj to null has exactly the same effect.

4.5 Binary operators

Binary operators have two operands: left and right operand.

Operator	Data types	Can overload	Description
*	number	YES	Multiplication
/	number	YES	Division
%	number	YES	Remainder
+	number, string, array	YES	Addition
-	number	YES	Substraction
<<	number	YES	Shift left
>>	number	YES	Shift right
<	number, string	YES	Less than
<=	number, string	YES	Less or equal than
>	number, string	YES	Greater than
>=	number, string	YES	Greater or equal than
==	all	YES	Logical equal
!=	all	YES	Logical not equal
&	number	YES	Bitwise and
|	number	YES	Bitwise or
∧	number	YES	Bitwise xor
&&	all	YES	Logical and
||	all	YES	Logical or
??	all		null-coalescing operator
=	all	YES	Assignment
+=	all	YES	Increment by
-=	number	YES	Decrement by
*=	number	YES	Multiply by
/=	number	YES	Divide by
%=	number	YES	Remainder of division by
=&	all		Assignment (no overloading)
&=	number	YES	Bitwise and with assigment
∧=	number	YES	Bitwise xor with assigment
|=	number	YES	Bitwise or with assignment
<<=	number	YES	Shift left with assignment
>>=	number	YES	Shift right with assigment
[ ]	string, array	YES	Index selector
.	class		Member selector
->	class		Member selector
::	class		Static member selector

Operator precedence:

1. ( )
2. new, delete, typeof, classof, length, value, !, , ++, –, (-/+ used as signs)
3. /, %, *
4. +, -
5. <<, >>
6. <, >, <=, >=
7. ==, !=
8. &, ∧, |, &&, ||, ??, =>(key-value specifier for arrays)
9. =, =&, +=, -=, /=, %=, *=, &=, ∧=, |=, <<=, >>=

Note that operators on the same level are evaluated in the order they are encountered, if no ( ) are used. If you group operations in parenthesis ( ), then you can alter the precedence. For example *2 + 2 * 2* will be evaluated at *6*. However, *(2 + 2) * 2* will be evaluated as 8. The + operator can be used on numbers, strings and arrays. **plusExample.con**

```
class Main {
    Main() {
        echo 1+1;
        echo "\n";
        echo "a"+"b";
        echo "\n";
        echo [1, 2, 3]+[4, 5];
    }
}
```

Outputs:

```
2
ab
Array {
        [0] => 1
        [1] => 2
        [2] => 3
        [3] => 4
        [4] => 5
}
```

You can also use + with operands of different types. The result will have the type of the left operand.

plusExample2.con

```
class Main {
    Main() {
        // will be evaluated as 1 + 2
        echo 1+"2";
        echo "\n";
        // will be evaluated as "2" + "1"
        echo "2"+1;
        echo "\n";
        echo [1, 2, 3] + 4;
    }
}
```

Outputs:

```
3
21
Array {
        [0] => 1
        [1] => 2
        [2] => 3
        [3] => 4
}
```

Note that if the array uses keys, the elements with keys are ignored in order to avoid key conflicts. For /, %, /= and %= ensude that the right operand is not zero. When running in interpreted mode you will get a runtime error. But when running in JIT, the expression will be evaluated to +/-INFINITE. The logical operators ||(OR) and &&(AND) have a different behavior. For ||, if the left operand is true, the second one is not evaluated. For &&, if the left operator is false, the second one is not evaluated. || and && accept any type of variable.

Delegate and class variables are evaluated to true. Arrays and strings are evaluated to true if they have at least one element, respectively character. Numbers are evaluated to true if they are not-zero.

The =& operator is identical with =, except that it cannot be overloaded.

equalExample.con

```
class Test {
    operator =(val) {
        echo "New value: $val";
    }
}

class Main {
    Main() {
        var obj=new Test();
        obj=null;
        obj=&null;
    }
}
```

Outputs:

```
New value: 0
```

The line obj=null calls the operator implemented in class Test. However, the next line, when the =& is used, the obj is set to null (as it would be deleted). The ?? operator (null-coalescing operator) evaluates to the right operand, if left operand is not true. The result can have either the type of left or right.

```
var result = a ?? b;
```

is exactly the same as

```
if (a)
    result = a;
else
    result = b;
```

The member selector operator . is identical with ->. Also, . can be used as an alias for ::. In practice I would recommend you use . in both situations, like this:

selectorsExample.con

```
class Test {
    var a;

    static function foo() {
    }
}

class Main {
    Main() {
        var obj=new Test();
        // the same as obj->a=10;
        obj.a=10;
        // the same as Test::foo();
        Test.foo();
    }
}
```

I would recommend using ".", for purely aesthetic reasons. Is up to you which one you prefer. The only exception is when calling an overriden member (see Virtual members) when it may cause some ambiguity.

The index operator [ ] can be used both in strings and arrays. The index, if a number, must be non-negative. Negative indexes will generate a run-time error. For arrays, requesting an index for a non-existing element, will cause the element creation and array length increase accordingly. For string, requesting an out-of-bounds character will return an empty string. If the index is a key, for arrays will be treated as a key, instead of an index. For strings, it will be evaluated to a number, and then the character on the position described by the string will be returned.

An array could have both key-value pairs and values. If a key is added, the given element can be accessed by position. Each new key creates an index equal with the previous length of the array. For example, adding "key 1" in an empty array, will generate index 0 for that key. The same key added in an array with 10 elements, will generate index 10.

```
var arr=new [];
arr["my key"]="my value";
arr[1]="Another value";

echo arr[0];
```

This will print “**my value**” on the console.

As a thing to remember: avoid non-integer indexes. In practice you should never use a index like 1.7. However, the Concept Interpreter will floor this to 1 and will return the element on the first position. The problem arise when running native code. Some optimization in the core, rounds 1.7 to 2. The same code, in this situation, can result in different results returned when running in interpreter and in JIT. As I said before, you should have no reason to use real indexes, and it is “bad programming”.

<, <=, >and >= can be used both for numbers and strings.

```
echo 1 < 2;
```

will print “**1**” (true).

```
echo "Andrew" < "Maria";
```

will also print “**1**” (true), because “A” precedes “M”. Note that string comparisons are case-sensitive. If we would compare “Andrew” with “Anna”, The difference will be made by the third letter, first two being the same.

4.6 Loops and conditions

In Concept, we have two methods of testing a condition.

1. Using **if..else**
2. Using **switch..case..default**

if evaluates a single expression enclosed by ( ). The expression can have any result type.

if (object)
is always true

if (delegate)
is always true

if (array)
is true if an array has at least one element. Is equivalent with if (array && length array)

if (string)
is true if a string has at least one character. Is equivalent with if (string && length string)

if (number)
is true if number is not zero

ifExample.con

```
class Main {
    Main() {
        var i=0;
        if (i==1)
            echo "i is 1";
        else
            echo "i is not 1";
    }
}
```

outputs

```
i is not 1
```

Note that if you have more than one statements in a if or else branch, you must group them using { }.

ifExample2.con

```
class Main {
    Main() {
        var arr = [1, 2, 3];
        if (arr) {
            echo "The sum is: ";
            echo arr[0] + arr[1] + arr[2];
        } else
            echo "arr is null";
    }
}
```

outputs

```
The sum is 6
```

You can put multiple if/else one after another.

```
if (i==1)
    echo "I is one";
else
if (i==2)
    echo "I is two";
else
if (i==3)
    echo "I is three";
else
    echo "I is not one, two nor three";
```

This can also be done by using **switch** statement. You can put multiple if/else one after another.

ifExample2.con

```
class Main {
    Main() {
        var i=2;
        switch (i) {
            case 1:
                echo "I is one";
                break;
            case 2:
                echo "I is two";
                break;
            case 3:
                echo "I is three";
                break;
            default:
                echo "I is not one, two, or three";
        }
    }
}
```

will output

```
I is two
```

Note the **break** after each case code. This is in order to prevent execution of the next branch. The **default** is executed if no other case branch is executed.

```
switch (i) {
    case 1:
        echo "I is one";
    case 2:
        echo "I is two";
    case 3:
        echo "I is three";
    default:
        echo "I is not one, two, or three";
}
```

will output

```
I is two
I is three
I is not one, two, or three
```

Case values can be anything (even expressions). Unlike C/C++ case, is not limited to constants. Case labels should be unique, but the concept core does not explicitly check for this, because labels are not limited to constants (like in C).

```
switch (expr) {
    case "a string":
        echo "expr is a string";
        break;
    case 2:
        echo "expr is two";
        break;
    case 2+1:
        echo "I is three";
        break;
}
```

Note that if i wold be 0(null, false), the first branch of the switch will be considered true, because i is a number of zero value, and ”a string” would evaluate to 0 as a number.

Concept also uses three types of loops:

1. **while**
2. **do .. while**
3. **for**

Keep in mind that a program spends most of its time in loops, and I recommend you to write carefully your loops.

while loops are initial-test loops, meaning that it will be executed only if the given condition is true.

whileExample.con

```
class Main {
    Main() {
        var i=10;
        while (i--) {
            echo i;
            echo " ";
        }
    }
}
```

outputs

```
9 8 7 6 5 4 3 2 1 0
```

When i becomes 0, it will be evaluated to false and the loop will end.

do..while are final-test loops. The same example modified to a end-loop test.

dowhileExample.con

```
class Main {
    Main() {
        var i=10;
        do {
            echo i;
            echo " ";
        } while (i--);
    }
}
```

outputs

```
10 9 8 7 6 5 4 3 2 1 0
```

Note that the test being done at the end, we have one more iteration. End-test are executed at least once.

for loops are similar with while loops, but have three parts: initialization expression, loop condition, and increment expression.

```
class Main {
    Main() {
        for (var i=0;i<10;i++) {
            echo i;
            echo " ";
        }
    }
}
```

outputs

```
0 1 2 3 4 5 6 7 8 9
```

You can have multiple initializations or increment expressions separated by comma.

```
class Main {
    Main() {
        for (var i=0, var j=0;i<10 && j<5;i++,j+=2) {
            echo i;
            echo " ";
        }
    }
}
```

outputs

```
0 1 2
```

As a note, if you want to initialize an array to a specific value, you can use:

```
for (var i=0;i<arr;i++) {
    arr[i]=true;
```

This type of loop is detected by the JIT optimizer, and is executed very efficient.

Regardless its type, a loop can be interrupted by the use of the **break** keyword. Also, can skip the rest of the loop inner code by the use of the **continue** keyword.

```
class Main {
    Main() {
        var i=10;
        while (true) {
            echo i;
            echo " ";
            i--;
            if (!i)
                break;
        }
        echo "\n";
        for (var j=0;j<10;j++) {
            if (j % 2)
                continue;
            echo j;
            echo " ";
        }
    }
}
```

outputs

```
10 9 8 7 6 5 4 3 2 1
0 2 4 6 8
```

Note that in the for-loop we skip the odd numbers and continue to next iteration.

4.7 Exceptions

Concept exceptions are similar with those used by C++ and Java, with the difference that they are not defined by data type. Constructors and destructors are not allowed to throw exceptions because the core will end up with a partially initialized object.

If you don't catch an exception in the calling function, it will be re-thrown until caught. If not caught, will cause program termination with an un-caught exception run-time error.

ExceptionExample.con

```
class Main {
    foo(i) {
        if (i==0)
            throw "Division by zero";
        return 1/i;
    }

    Main() {
        try {
            foo(0);
        } catch (var Exception) {
            echo Exception;
        }
    }
}
```

outputs

```
Division by zero
```

In practice, you will probably encapsulate exceptions in objects.

```
class MyException {
    var Code;
    var Description;

    MyException(code, description) {
        this.Code = code;
        this.Description = description;
    }
}
[...]
    try {
        [...]
        throw new MyException(1, "Division by zero");
        [...]
    } catch (var Exception) {
        echo Exception.Description;
    }
[...]
```

If foo throws an exception being called by foo2, which doesn't catch the exception, and foo2 is called by foo3 which has a try/catch block eclosing the call to foo2, the exception will end up in foo3's catch block.

Each try must have exactly one corresponding catch block. You cannot have multiple catch blocks.

4.8 Include and import

A project will have many sources and will use many modules. Note that *include*, *import* and *define* can be used only outside class definition. *define* will create a compile-time constant that will be replaced by the parser with the given value.

For including another source, you may use the **include** statement like this:

Test.con

```
define SOME_CONSTANT  10

class Test {
    var SomeMember;
}
```

TestInclude.con

```
include Test.con

class Main {
    Main() {
        var t=new Test();
        t.SomeMember = SOME_CONSTANT;
    }
}
```

import is similar with include, but instead of Concept sources, it imports binary modules.

TestImport.con

```
import standard.lib.str

class Main {
    Main() {
        echo ToUpper("The quick brown fox jumps over the lazy dog");
    }
}
```

Will output:

```
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
```

The ToUpper function is a native function written in C and defined in standard.lib.str. It takes one parameter, a string, and returns the a new string containing the uppercase version of the given parameter.

Chapter 5

Classes and objects

Classes actually define data structures with or without various operations attached. A class itself does not contain any data. Is more like a specification of a data type.

Concept is a strict object-oriented languages, meaning that every function or variable must be part of a class.

Classes are declared using the "**class**" keyword. Objects are created from classes using the "**new**" operator.

A class may have a constructor and a destructor. The constructor is called when you instantiate the class using "new", and the destructor is called just before the memory associated with the object is freed. Remember that both constructors and destructors are not allowed to throw exceptions. You can't have two ore more members with the same name in the same class. You can override a member by using the "override" keyword (will be discussed in the next sections).

```
class A {
   A() {
      // nothing
   }
}

[...]
var a=new A();
```

```
[...]
```

In the given example, *a* is an object and *A* is a class.

5.1 Member variables

A member variable is declared using the "**var**" keyword (short for *variant*). An array can be declared using "**var**[]". You can assign default static values to a class member. Static values are limited to strings and numbers. A default value must be a constant(no expressions allowed). For other data types (array, objects and delegates) you can assign default values in a constructor.

TestVarMember.con

```
class Person {
    var Name="Uknown";
    var[] SomeArray;
    var Age;
}

class Main {
    Main() {
        var p=new Person();
        echo "Default name: ${p.Name}\n";
        p.Name="Eduard";
        echo "New name: ${p.Name}\n";
    }
}
```

Will output:

```
Default name: Unknown
New name: Eduard
```

5.2 Function members

Function members are declared using the optional keyword “**function**”. It can have a maximum of 65535 parameters. Each parameter may have a default value, which must be a static constant(string or number). Note that after a member that has a default value, all the following members must have default values.

FunctionMemberExample.con

```
class Main {
    function foo(a, b, c, d=4, e=5) {
        echo "$a, $b, $c, $d, $e\n";
    }

    function Main() {
        this.foo(1, 2, 3);
    }
}
```

Will output:

```
1, 2, 3, 4, 5
```

Also, function member can have type validation. Type validations can be ”number”, ”string”, ”array”, ”delegate”, ”object” or a class name.

A function can return any kind of variable using the “**return**” keyword. If a function has no *return* statement, it will automatically return zero (null/false).

FunctionMemberExample2.con

```
class SomeClass {
    var SomeValue=1;
}

class Main {
    foo(SomeClass a, string b, number c, object d, array e) {
        var result=a.SomeValue + b + c + d.SomeValue + e[0];
        return result;
    }
    Main() {
        var d=new SomeClass();
        d.SomeValue=4;

        echo "Result is: "+this.foo(new SomeClass(), "2", 3, d, [5]);
    }
}
```

Will output:

```
Result is: 15
```

Note that in the previous example, the **function** keyword was omitted. A member with no other specifier is assumed to be a function.

You may have referenced parameters (or OUT parameters) if you use the "**var**" prefix. Also, when a member is called within it's own class, you may call it without the "this" reference(instead of *this.foo* you can simple use *foo*). Note that if you have a local variable called foo, then this would be mandatory, because local variable are checked first by the compiler.

FunctionMemberExample3.con

```
class Main {
    foo(a, var b, c) {
        a++;
        b++;
        c++;
    }

    Main() {
        var a=1, b=2, c=3;
        foo(a, b, c);
        echo "A is $a, B is $b, C is $c";
    }
}
```

Will output:

```
A is 1, B is 3, C is 3
```

In **foo**, all the parameters are incremented, but only *b* has the *var* specifier. After calling foo, you noticed that only b has a new value. *a* and *c* where copied in foo, but _ was referenced, so any changed made in foo, was reflected in the calling function.

Note that array, objects and delegates are not copied. When an object is passed as a parameter, the variable is copied, but the variable itself is a reference to the object.

5.3 Constructor and destructor

Constructors and destructors are special functions that are called automatically when creating, respectively destroying an object. A class can have only one constructor and/or only one destructor.

constructorExample.con

```
class A {
    var SomeValue;

    A(number a_value) {
        this.SomeValue=a_value;
    }
}

class Main {
    Main() {
        var a=new A(10);
    }
}
```

Creating object *a* will automatically call *A::A(number a_value).* Remember that a constructor is not allowed to throw exception.

Destructors are special functions named “finalize”. A destructor cannot take parameters and is not allowed to throw an exception. Also, there is no explicit guarantee that it will be called, like the case of the constructor.

destructorExample.con

```
class A {
    doSomethingWithA() {
        echo "Do something with a\n";
    }

    finalize() {
        echo "A is about to be destroyed";
    }
}

class Main {
    Main() {
        var a=new A();
        a.doSomethingWithA();
    }
}
```

Output:

```
Do something with a
A is about to be destroyed
```

5.4 Properties

Properties are special members declared with **property** keyword that have two methods - **get** and **set**. A property may be read-only or read-write. A property is useful when you want to implement validations to a member variable.

propertyExample.con

```
class A {
    var _SomeValue;
    property SomeValue { get _SomeValue, set SetSomeValue }

    function SetSomeValue(number val) {
        if (val==0)
            throw "SomeValue cannot be zero";
        this._SomeValue=val;
    }
}

class Main {
    Main() {
        var a=new A();
        try {
            a.SomeValue=1;
            echo a.SomeValue;
            echo "\n";
            a.SomeValue=0;
        } catch (var Exception) {
            echo Exception;
        }
    }
}
```

Output:

```
1
SomeValue cannot be zero
```

get statament is mandatory for every property, but *set* is optional. A property that has no *set* method, it is called a read-only property.

Both *get* and *set* can reference either a member variable or a function. It cannot reference another property.

5.5 Access control

Class members(variables, functions and properties) can have three access specifiers:

public

the member is accessible by any other objects. This is the default acces (if not specified, *public* is to be assumed)

private

the member is accessible only by the same class

protected
the member is accessible only by the same class and its subclasses

accessExample.con

```
class A {
   private var _SomeValue;
   public property SomeValue { get _SomeValue, set SetSomeValue }

   private SetSomeValue(number val) {
      this._SomeValue=val;
   }
}

class Main {
   Main() {
      var a=new A();
      a.SomeValue=1;
   }
}
```

In this example, the access to *_SomeValue* is mediated by the public SomeValue property. If we try to execute a._SomeValue=1 from the Main class, you will get a run-time error.

```
In accessExample.con:
  In class A:
    In member A.A:
      0. E190 on line 14: You can't access a private member of a
           class ('_SomeValue')
There are RUN-TIME errors.
```

protected access a combination of private and public. The member is accessible for subclasses, but private for any other class. You will learn in the next sections about inheritance.

protectedExample.con

```
class A {
   protected var _SomeValue;
   public property SomeValue { get _SomeValue, set SetSomeValue }
```

```
    private SetSomeValue(number val) {
        this._SomeValue=val;
    }
}

class B extends A {
    public AccessTest(number val) {
        this._SomeValue=val;
    }
}

class Main {
    Main() {
        var a=new B();
        a.AccessTest(1);
    }
}
```

The value of *_SomeValue* will be 1. *class B* is able to access the protected member *_SomeValue* defined in *class A* because B is a child of A. However, class Main can't access it.

5.6 Static members

Static functions are not linked with an actual object. Only functions can be declared static, and the use of **this** is not allowed. Everything else works just like an ordinary function. You can even set access control for a static function.

staticExample.con

```
class A {
    static function Print(string message) {
        echo message;
    }
}

class Main {
    Main() {
        A.Print("Hello world!");
        // exactly the same as
        // A::Print("Hello world!");
    }
}
```

outputs:

```
Hello world!
```

Note that you can use :: instead of . when dealing with static members.

Note that you didn't have to create an object of type A. If you use *var a=new A()* you will be able to call *Print* like a standard member function - *a.Print(..)*.

Static functions cannot be referenced by delegates, unless you associate the function with an actual object. It would be illegal in the above example to set *var d=A.Print*. Instead you can achieve this this way:

```
[...]
    static function Print(string message) {
        echo message;
    }
[...]
    var a=new A();
    var deleg=a.Print;
[...]
```

5.7 Virtual members and overriding

In Concept, any class member can be overridden, implicitly making it virtual. We will call virtual members, the functions without body (pure-virtual). There are two ways of implementing this. You can define a function without a body, and then, in a subclass, you can override and implement that method, by using the **override** keyword.
virtualExample.con

```
class A {
    function SomeTest();
}
```

```
class B extends A {
    override SomeTest;
    function SomeTest() {
        echo "Some test";
    }
}

class Main {
    Main() {
        var b=new B();
        b.SomeTest();
    }
}
```

In this case, *A.SomeTest* is a pure virtual function. However, A’s *SomeTest* may have actual code.

virtualExample2.con

```
class A {
    function SomeTest() {
        echo "A’s some test";
    }
}

class B extends A {
    override SomeTest;
    function SomeTest() {
        echo "B’s some test\n";
        // call previous function
        A::SomeTest();
    }
}

class Main {
    Main() {
        var b=new B();
        b.SomeTest();
    }
}
```

Outputs:

```
B’s some test
A’s some test
```

Note that when referencing a previous overriden member, in a class that has an implemented constructor, “::” is mandatory for avoiding member/class auto-reference ambiguity. If “.” would be used, the Concept Parser will resolve the class name to its constructor, assuming that we want to call or reference it.

A.SomeTest is not a static function call. Is a call to the previous implementation of *SomeTest*, the one in class *A*. Note, you can use :: instead of . in *A::SomeTest()*. The call to the previous implementation can be made from any function defined in B or it’s subclasses.

Notes that when overriding constructors and destructors, you must explicitly call the previous implementation, if needed.

The second method of creating virtual function is by using the **event** ... **triggers** construct. This creates an alias for a function that must be implemented in a subclass.

virtualExample3.con

```
class A {
   event SomeTest triggers SomeTestImplementation;
}

class B extends A {
   function SomeTestImplementation() {
      echo "SomeTestImplementation\n";
   }
}

class Main {
   Main() {
      var b=new B();
      b.SomeTest();
   }
}
```

Outputs:

```
SomeTestImplementation
```

When calling *SomeTest*, defined in *A*, the core will look for a member called *SomeTestImplementation*. If the member is implemented, then it will be executed.

When using this kind of alias, is not mandatory for the trigger function to be implemented. If the trigger function is not implemented, nothing will happen, assuming that is called without any parameter. If the function takes parameters, you will have a parameter count run-time error when SomeTestImplementation is not implemented.

5.8 Operator overloading

Most of the Concept operators can be implemented in classes. Refer to sections "*Unary operators*" and "*Binary operators*" to see which one can be implemented.

An operator can be defined using the "**operator**" keyword. Operators obey the same rules as any other function, except that binary operators must take exactly one parameter and unary operators must take no parameters. You may also use access control with an operator making it private, protected or public. By default they are public, like any other member.

operatorExample.con

```
class Integer {
    private var _i;

    Integer(number val=0) {
        _i=val;
    }

    operator+(a) {
        echo "Using overloaded operator\n";
        var ival;
        switch (typeof a) {
            case "number":
                ival=a;
                break;
            case "string":
                ival=value a;
                break;
            case "array":
                ival=a[0];
                break;
            case "class":
                if (classof a=="Integer") {
                    ival=a._i;
                    break;
                }
            default:
                throw "Invalid data type";
        }
        return new Integer(ival + this._i);
    }

    operator!() {
        if (!_i) {
            echo "NOT Integer is true\n";
            return true;
        }
        echo "NOT Integer is false\n";
        return false;
    }

    ToNumber() {
        return _i;
    }
}

class Main {
    Main() {
        var a=new Integer(1);
        var b=new Integer(2);
        var c=a+b;
        echo "Result is: "+c.ToNumber()+"\n";
        echo "Result of !c: "+!c;
    }
}
```

Outputs:

```
Using overloaded operator
Result is: 3
NOT Integer is false
Result of !c: 0
```

The =& (reference assignment) cannot be overloaded, in order for the programmer to have a mechanism of forcing a reference assignment.

5.9 Duck typing

In computer programming with object-oriented programming languages, duck typing is a style of typing in which an object's methods and properties determine the valid semantics, rather than its inheritance from a particular class or implementation of a specific interface. The name of the concept refers to the duck test, attributed to James Whitcomb Riley, which may be phrased as follows:

When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.

In duck typing, one is concerned with just those aspects of an object that are used, rather than with the type of the object itself. For example, in a non-duck-typed language, one can create a function that takes an object of type Duck and calls that object's walk and quack methods. In a duck-typed language, the equivalent function would take an object of any type and call that object's walk and quack methods. If the object does not have the methods that are called then the function signals a run-time error. If the object does have the methods, then they are executed no matter the type of the object, evoking the quotation and hence the name of this form of typing.

Duck typing is aided by habitually not testing for the type of arguments in method and function bodies, relying on documentation, clear code and testing to ensure correct use.

duckExample.con

```
class Duck {
    quack() {
        echo "Quaaaaaack!";
    }

    feathers() {
        echo "The duck has white and gray feathers.";
    }
}

class Person {
    var name="John Smith";

    quack() {
        echo "The person imitates a duck.";
    }

    feathers() {
        echo "The person takes a feather from the ground and shows
            it.";
    }
}

class Main {
    in_the_forest(duck) {
        duck.quack();
        echo "\n";
        duck.feathers();
    }

    Main() {
        var donald=new Duck();
        var john=new Person();
        in_the_forest(donald);
        echo "\n";
        in_the_forest(john);
    }
}
```

Outputs:

```
Quaaaaaack!
The duck has white and gray feathers.
The person imitates a duck.
The person takes a feather from the ground and shows it.
```

5.10 Inheritance - simple and multiple

A Concept class can inherit one or more base classes using the “**extends**” keyword. All Concept classes are inheritable. **inheritanceExample.con**

```
class Vegetable {
    public var edible;
    public var color;

    Vegetable(edible,color) {
        this.edible = edible;
        this.color = color;
    }
}

class Spinach extends Vegetable {
    public var cooked;

    Spinach() {
        this.cooked=0;
        // call previous constructor
        this.Vegetable(1,"green");
    }

    CookIt() {
        this.cooked = 1;
    }
}

class Main {
    Main() {
        var test=new Spinach();
        echo "---- Vegetable and Spinach ---\n";
        echo "Color : "+test.color+"\n";
        var edible=test.edible;

        if (!edible)
            edible="no";
        else
            edible="yes";

        echo "Edible : "+edible+"\n";
        echo "-------- Only Spinach --------\n";
        var cooked=test.cooked;

        if (!cooked)
            cooked="no";
        else
            cooked="yes";

        echo "Cooked : "+cooked+"\n";
        echo "Cooking ...\n";

        test.CookIt();
        cooked=test.cooked;

        if (!cooked)
            cooked="no";
        else
            cooked="yes";

        echo "Cooked : "+cooked;
    }
}
```

Outputs:

```
---- Vegetable and Spinach ---
Color  : green
Edible : yes
-------- Only Spinach --------
Cooked : no
Cooking ...
Cooked : yes
```

Note that when in single class inheritance, you may use the reserved word *super* instead of *this.Vegetable* to reference the superclass' constructor.

```
Spinach() {
    // call the previous constructor
    super(1,"green");
    this.cooked=0;
}
```

A Concept class can inherit more than one superclass. It can't inherit two classes, each of them having a member with a name that can be found in the other class, because it will generate a naming conflict(this will generate compile-time error).

```
class A {
    A() {
        echo "A";
    }
}

class B {
    B() {
        echo "B";
    }
}

class C extends A extends B {
    C() {
        this.A();
        this.B();
        echo "C";
    }
}

class Main {
    Main() {
        var c=new C();
    }
}
```

Outputs:

```
ABC
```

5.11 Anonymous functions

An anonymous function, also called a lambda function, are a form of nested function, that have no name. In reality, the Concept Core will create a standard function, that will be executed by the Concept Core like a typical function. Its access rights will be the same with the function defining the lambda function.

There are two types of lambda functions: with parameters or without any input parameters. Note that these function cannot access the variables defined in the defining function. A lambda function definition will always create a delegate variable.

A parameterless lambda function can be defined as:

LambdaTest.con

```
class Main {
    Main() {
        var foo = { echo "Hello World!"; };
        foo();
    }
}
```

When parameters are needed, it can be defined using the **function** keyword:

LambdaTest2.con

```
class Main {
    Main() {
        var foo = function(msg) { echo msg; };
        foo("Hello world!");
    }
}
```

A Concept anonymous function cannot access local variables, but can access object variables. In the previous example, assuming that Main is not defined static (case being, all the lambda function will be considered static), the lambda functions will be able to access any of the *Main* members.

LambdaTest3.con

```
class Main {
    var msg = "Hello World!";

    Main() {
        var foo = { echo this.msg; };
        foo();
    }
}
```

Anonymous function have different names on various programming languages. You may know them as closure, blocks, inline agents or function objects.

This is a relatively new feature for Concept being first available on version 2.8. However, code compiled with a 2.8 compiler using lambda functions will run with no problem on any previous version of Concept.

Chapter 6

Static functions

Concept core is able to run any native function. Native functions can be written in languages like C and C++. These are the only functions that are not bound to a class. Concept uses an *Invoke* function that enables the called function to communicate with the core. In the Concept distribution package you will find two files - *stdlibrary.h* and *stdlibrary.cpp*. These files define all the structures and macros you will need to create a static function.

The Invoke function has the following prototype:

INTEGER **Invoke**(INTEGER INVOKE_TYPE, ...);

For each INVOKE_TYPE you have a list of parameters:

INVOKE_SET_VARIABLE (void *variable, INTEGER type, char *str, NUMBER n)
Sets a variable type and value

INVOKE_GET_VARIABLE (void *variable, INTEGER *type, char **str, NUMBER *n)
Gets a variable type and value

INVOKE_SET_CLASS_MEMBER (void *object, char *mname, INTEGER type, char *str, NUMBER n)
Sets value and type for a class member (*var*)

INVOKE_GET_CLASS_MEMBER (void *object, char *mname, INTEGER *type, char **str, NUMBER *n)
Gets value and type for a class member (*var*)

INVOKE_GET_CLASS_VARIABLE (void *variable, char *mname, void **membervariable)
Gets a variable from an object as a handle

INVOKE_FREE_VARIABLE (void *variable)
Reduces the link count for a variable by 1, and frees the memory if it reaches 0

INVOKE_CREATE_ARRAY (void *variable)
Creates an array in the specified variable

INVOKE__ARRAY_VARIABLE (void *variable, INTEGER index, void **var_element)
Gets a handle to a variable from an array by index

INVOKE_ARRAY_VARIABLE_BY_KEY (void *variable, char *key, void **var_element)
Gets a handle to a variable from an array by key

INVOKE_GET_ARRAY_ELEMENT (void *variable, INTEGER index, INTEGER *type, char **str, NUMBER *n)
Gets an array element by index

INVOKE_SET_ARRAY_ELEMENT (void *variable, INTEGER index, INTEGER type, char *str, NUMBER n)
Sets an array element by index

INVOKE_GET_ARRAY_COUNT (void *variable)
Returns the array element count

INVOKE_GET_ARRAY_ELEMENT_BY_KEY (void *variable, char *key, INTEGER *type, char **str, NUMBER *n)
Gets an array element by key

INVOKE_SET_ARRAY_ELEMENT_BY_KEY (void *variable, char *key, INTEGER type, char *str, NUMBER n)
Sets an array element by index

INVOKE_CALL_DELEGATE (void *var_delegate, void **result, void **exception, INTEGER param_count>=0, [INTEGER type, char

*str, NUMBER n,]). It can be called using a null-terminated array of variables (void *var_delegate, void **result, void **exception, INTEGER param_count=-1, void *parameters[], (void *)NULL);
Calls a delegate (received as a parameter)

INVOKE_COUNT_DELEGATE_PARAMS (void *var_delegate)
Returns the parameters count of a delegate

INVOKE_LOCK_VARIABLE (void *variable)
Increments the link-count for a variable. Use free variable for decrementing the links

INVOKE_GET_ARRAY_KEY (void *arr_variable, INTEGER index, char **key)
Gets the key for the specified index of an array

INVOKE_GET_ARRAY_INDEX (void *arr_variable, char *key, INTEGER *index)
Gets the index for the specified key

INVOKE_CREATE_VARIABLE (voir **variable)
Creates a Concept variable

INVOKE_DEFINE_CONSTANT (CONTEXT, char *name, char *value)
Defines a Constant. This function can be called only from ON_CREATE_CONTEXT (before the program is actually executed)

INVOKE_DEFINE_CLASS (CONTEXT, char *class_name, char *member_name1, ..., char *member_nameN, (char *)NULL)
Defines a class

INVOKE_CREATE_OBJECT (CONTEXT, void *variable, char *class_name)
Creates an object

INVOKE_GET_MEMBER_FROM_ID (void *object_ref, intptr_t member_id, char **member_name)
Gets a class member from an ID

INVOKE_DYNAMIC_LOCK (void *variable)
Increases the class/delegate/array links count, without increasing the variable link count

INVOKE_HAS_MEMBER (void *object_ref, char *member_name)
Check if class's object has a specific member

INVOKE_OBJECT_LINKS (void *variable)
Returns the object link count

INVOKE_VAR_LINKS (void *variable)
Returns the variable link count

INVOKE_CLI_ARGUMENTS (intptr_t *argc, char ***argv)
Gets the command line arguments

INVOKE_SORT_ARRAY_BY_KEYS (void *in_array_object, void *out_sorted_array_object)
Creates a new array sorted by the given array keys

INVOKE_CHECK_POINT (int seconds)
Sets the application operation timeout in seconds. If set to 0, the timeout will be infinite

INVOKE_ARRAY_ELEMENT_IS_SET (void *variable, INTEGER index, char *key)
Check if an array element is set. If index >= 0, then the index will be checked, else, if index <0, the key will be used

INVOKE_GET_DELEGATE_NAMES (void *delegate_variable, char **class_name, char **member_name)
Gets the delegate class name and member name

INVOKE_GET_USERDATA (CONTEXT, void **userdata)
Gets the user data associated with the core

INVOKE_GET_THREAD_DATA (void **data)
Get the data associated with the thread

INVOKE_SET_THREAD_DATA (void *data)
Set the data associated with the thread

INVOKE_NEW_BUFFER (INTEGER size, char **buffer)
Allocates a new core buffer

INVOKE_DELETE_BUFFER (char *buffer)
Deletes a core buffer

INVOKE_PRINT (CONTEXT, char *what, INTEGER what_len)
Outputs to stdout a string.

INVOKE_CALL_DELEGATE_THREAD (void *var_delegate, void **result, void **exception, INTEGER param_count>=0, [INTEGER type, char *str, NUMBER n,]). It can be called using a null-terminated array of variables (void *var_delegate, void **result, void **exception, INTEGER param_count=-1, void *parameters[], (void *)NULL);
Calls a delegate in a new thread

INVOKE_CALL_DELEGATE_THREAD_SAFE (void *var_delegate, void **result, void **exception, INTEGER param_count>=0, [INTEGER type, char *str, NUMBER n,]). It can be called using a null-terminated array of variables (void *var_delegate, void **result, void **exception, INTEGER param_count=-1, void *parameters[], (void *)NULL);
Calls a delegate in a new thread, using semaphores for safe call

INVOKE_CREATE_DELEGATE (void *class_variable, void *delegate_variable, char *member_name)
Creates a delegate

INVOKE_FREE_VARIABLE_REFERENCE (void *variable)
Frees a variable regarding of its link count. *WARNING: you should avoid using this function*

INVOKE_THREAD_LOCK (void *class_or_delegate_variable)
Locks/unlock an internal semaphore used for synchronizing threads

INVOKE_EXTERNAL_THREADING (void *class_or_delegate_variable)
Increments the core thread count reference

Variable type can be: VARIABLE_NUMBER, VARIABLE_STRING, VARIABLE_CLASS, VARIABLE_ARRAY or VARIABLE_DELEGATE

Invoke will return a value equal or greater than zero if succeeded, or a negative value if it failed. The returned errors are CANNOT_INVOKE_INTERFACE(-10), INVALID_INVOKE_PARAMETER(-20) or CANNOT_INVOKE_IN_THIS_CASE(-30).

A C function wrapper must contain NULL in case of success or a human-readable, zero-terminated string describing the error in case of error.

6.1 Mapping a simple C function

A set of macros defined in *stdlibrary.h* helps you map with minimum effort a C function. For that, you must create two files: a .h (header file) and a .cpp (C++ source file).

All Concept modules must use the C calling convention.

The *CONCEPT_FUNCTION(function_name)* macro it's a convenient way to declare a function in a header file. In the source file you must use CONCEPT_FUNCTION_IMPL(function_name, parameters_count), CONCEPT_FUNCTION_IMPL_VARIALBE_PARAMS(function_name, min_parameters_count) or CONCEPT_FUNCTION_IMPL_MINMAX_PARAMS(function_name, min_parameters_count, max_parameters_count).

Note that concept function are prefixed by the "*CONCEPT_*" prefix in order to avoid conflicts with existing C functions. The native name for the *CONCEPT_FUNCTION(divide)* will be *CONCEPT_divide*.

library.h

```
#ifndef __LIBRARY_H
#define __LIBRARY_H

// provided with concept, defines all
// the macros and data structures
#include "stdlibrary.h"

extern "C" {
    // optional functions called automatically when library is
        loaded
    CONCEPT_DLL_API ON_CREATE_CONTEXT MANAGEMENT_PARAMETERS;
    CONCEPT_DLL_API ON_DESTROY_CONTEXT MANAGEMENT_PARAMETERS;

    CONCEPT_FUNCTION(divide)
    CONCEPT_FUNCTION(sum)
    CONCEPT_FUNCTION(stringTest)
}

#endif
```

main.cpp

```
#include "library.h"

#define SOME_CONSTANT 100

CONCEPT_DLL_API ON_CREATE_CONTEXT MANAGEMENT_PARAMETERS {
    // this is called when library is first loaded
    DEFINE_ECONSTANT(SOME_CONSTANT)
    DEFINE_SCONSTANT("SOME_STRING_CONSTANT", "Hello world!")
    DEFINE_ICONSTANT("SOME_INTEGER_CONSTANT", 1)
    DEFINE_FCONSTANT("SOME_FLOAT_CONSTANT", 1.2)
    return 0;
}

CONCEPT_DLL_API ON_DESTROY_CONTEXT MANAGEMENT_PARAMETERS {
    // this is called when library is unloaded
    return 0;
}

CONCEPT_FUNCTION_IMPL(divide, 2)
    T_NUMBER(0)
    T_NUMBER(1)

    if (PARAM(2)==0)
        return (void *)"divide: Division by zero";
    NUMBER res=PARAM(0)/PARAM(2);
    RETURN_NUMBER(res)
END_IMPL

CONCEPT_FUNCTION_IMPL_VARIABLE_PARAMS(sum, 0)
    double sum=0;
    for (int i=0;i<PARAMETERS_COUNT;i++) {
        T_NUMBER(i)
        sum+=PARAM(i);
    }

    RETURN_NUMBER(sum)
END_IMPL
```

```
CONCEPT_FUNCTION_IMPL(stringTest, 1)
    T_STRING(0)
    char *param=PARAM(0);
    int param_len=PARAM_LEN(1);

    RETURN_BUFER(param, param_len)
END_IMPL
```

We defined here two functions: divide(a, b), computes a/b and sum(...), computes p1+p2...pn.

After compiling with a C++ compiler, we can use the following Concept program to invoke these functions, assuming our library was compiled to mylibrary (see section Modules for more information).

The resulting mylibrary.dll or mylibrary.so must be placed either in the Concept Application Server *Library* directory or in your application's directory. Note that the core will search a library in *Library* first, and the in the current directory.

```
import mylibrary

class Main {
    Main() {
        echo "1/2 = " + divide(1,2) + "\n";
        echo "1+2+3+4+5 = " + sum(1,2,3,4,5) + "\n";
        echo stringTest(SOME_STRING_CONSTANT);
    }
}
```

will output

```
1/2 = 1.5
1+2+3+4+5 = 15
Hello world!
```

Note that RETURN_NUMBER, doesn't end the function execution. You must call *return 0* in order to end the execution. RETURN macros just set the result to the given value.

There are a few RETURN macros: RETURN_NUMBER(double), RETURN_STRING(char *), RETURN_BUFFER(char *, int len), RETURN_ARRAY(array_object). You can modify an input parameter by using SET_NUMBER(parameter_index, double), SET_STRING(parameter_index, char *), SET_BUFFER(parameter_index, char *, INTEGER len).

The macros T_NUMBER(parameter_index), T_STRING(parameter_index), T_ARRAY(parameter_index), T_OBJECT(parameter_index), T_DELEGATE(parameter_index) and T_HANDLE(parameter_index) perform a type check for the given parameter index, returning an error on type mismatch. Also, these macros create the PARAM(parameter_index) (and PARAM_LEN(parameter_index) for string parameters) access variables.

Note that when using RETURN_BUFFER, if len is 0, the given *char ** parameter must be null terminated. If you want to return a NULL string, RETURN_STRING("") is the correct way to do it.

6.2 Working with arrays

stdlibrary.h provides a few macros that help you dealing with arrays. For creating arrays, we have the CREATE_ARRAY(variable) macro. For example, if we want to return an array, we can use CREATE_ARRAY(RESULT). If we want to use a parameter as an out value, CREATE_ARRAY(PARAMETERS(0)) for parameter 1 (first parameter has 0 index, second 1).

library.h

```
#ifndef __LIBRARY_H
#define __LIBRARY_H

// provided with concept, defines all
// the macros and data structures
#include "stdlibrary.h"

extern "C" {
    CONCEPT_FUNCTION(CreateArrayByElements)
    CONCEPT_FUNCTION(AddVariableKey)
    CONCEPT_FUNCTION(CreateMatrix)
}

#endif
```

main.cpp

```
#include "library.h"

CONCEPT_FUNCTION_IMPL_VARIABLE_PARAMS(CreateArrayByElements, 0)
    CREATE_ARRAY(RESULT)
    for (int i=0;i<PARAMETERS_COUNT;i++) {
        T_NUMBER(i)
        Invoke(INVOKE_SET_ARRAY_ELEMENT, RESULT, (INTEGER)i,
            (INTEGER)VARIABLE_NUMBER, "", PARAM(i));
    }
END_IMPL

CONCEPT_FUNCTION_IMPL(AddVariableKey, 3)
    T_ARRAY(0)
    T_STRING(1)
    T_NUMBER(2)

    Invoke(INVOKE_SET_ARRAY_ELEMENT_BY_KEY, PARAMETER(0), PARAM(1),
        (INTEGER)VARIABLE_NUMBER, "", PARAM(2));
    RETURN_NUMBER(0)
END_IMPL

CONCEPT_FUNCTION_IMPL(CreateMatrix, 2)
    T_NUMBER(0)
    T_NUMBER(1)
    CREATE_ARRAY(RESULT)

    int lines = PARAM_INT(0);
    int columns = PARAM_INT(1)

    for (INTEGER i=0;i<lines;i++) {
        void *line;
        Invoke(INVOKE_ARRAY_VARIABLE, RESULT, index, &line);
        CREATE_ARRAY(line)

        for (INTEGER j=0;j<columns;j++) {
            Invoke(INVOKE_SET_ARRAY_ELEMENT, line, (INTEGER)i,
                (INTEGER)VARIABLE_NUMBER, "", (NUMBER)0);
        }
    }
END_IMPL
```

After compiling the library, under the name myarraylibrary(.dll/.so), you can call the functions using:

```
import myarraylibrary

class Main {
    Main() {
        var[] arr;
        var elements=CreateArrayByElements(1,2,3);

        AddVariableKey(arr, "Key 1", 1)
        AddVariableKey(arr, "Key 2", 2)

        var matrix=CreateMatrix(3, 3);
    }
}
```

elements will be an array containing [1, 2, 3]. *arr* will have [”Key 1”: 1, ”Key 2”: 2] and *matrix* will be a matrix of 3 by 3 containing zeros.

Note that you cannot remove elements from an array. If you need to delete an element from an array, you must create a new one and import all but the delete element from the initial array.

6.3 Working with objects

C static function can access concept data members, delegates and define new classes. Although is easily possible to create a class from a static C function, I wouldn’t recommend it, because the structure of the class won’t be visible to the programming.

library.h

```
#ifndef __LIBRARY_H
#define __LIBRARY_H
```

```
// provided with concept, defines all
// the macros and data structures
#include "stdlibrary.h"

extern "C" {
    CONCEPT_FUNCTION(CallDelegate)
    CONCEPT_FUNCTION(GetValueOf)
}

#endif
```

main.cpp

```
#include "library.h"

CONCEPT_FUNCTION_IMPL(CallDelegate, 2)
    T_DELEGATE(0)

    void *RES=0;
    void *EXCEPTION=0;
    void *delegate_PARAMS[2];
    delegate_PARAMS[0]=PARAMETER(1);
    delegate_PARAMS[1]=0;
    Invoke(INVOKE_CALL_DELEGATE, PARAMETER(0), &RES, &EXCEPTION,
        (INTEGER)-1, delegate_PARAMS);

    Invoke(INVOKE_FREE_VARIABLE, EXCEPTION);
    Invoke(INVOKE_FREE_VARIABLE, RES);
    RETURN_NUMBER(0)
END_IMPL

CONCEPT_FUNCTION_IMPL(GetValueOf, 2)
    T_OBJECT(0)
    T_STRING(1)

    void *member;
    if (!IS_OK(Invoke(INVOKE_GET_CLASS_VARIABLE, PARAM(0),
        PARAM(1), &member)))
        return (void *)"GetValueOf: Invalid member name";

    INTEGER member_type=0;
    char *member_szData;
    NUMBER member_nData;
    Invoke(INVOKE_GET_VARIABLE, member, &member_type,
        &member_szData, &member_nData);

    switch (member_type) {
        case VARIABLE_NUMBER:
            RETURN_NUMBER(member_nData)
            break;
        case VARIABLE_STRING:
            // for strings, member_nData contains the length
            RETURN_BUFFER(member_szData, member_nData);
            break;
        default:
            return (void *)"GetValueOf: result is neither a number
                nor a string";
    }
END_IMPL
```

Note that if you want to retain a reference to the delegate parameter, for calling it after the function execution ends, you must increase its reference count by using:

Invoke(INVOKE_LOCK_VARIABLE, PARAMETER(0))

This will ensure that the variable won't be freed by the time you need to call the delegate.

After compiling the library, under the name mydelegatelibrary(.dll/.so), you can call the functions using:

```
import myarraylibrary

class Main {
    var someMember="... Hello to you too";

    foo(msg) {
        echo msg;
        // get member by its name as a string
        echo GetValueOf(this, "someMember");
    }

    Main() {
        CallDelegate(this.foo, "Hello world!");
    }
}
```

Will output ”*Hello world! ... Hello to you too*”. *foo* is called from a C function invoked by *Main* (*CallDelegate*). Then, foo calls another C function that gets the value of “*someMember*” in the current object (*this*).

You can combine this functions anyway you like. Note that some APIs require an actual reference to an array or object. For example INVOKE_HAS_MEMBER requires a reference to the actual object instead of the variable holding that object. For that you can either use the GET_ARRAY(parameter_index, void *object), GET_OBJECT(parameter_index, void *object) (identical with GET_ARRAY) or GET_DELEGATE(parameter_index, void *object, double delegate_member) when referencing local parameters, or Invoke(INVOKE_GET_VARIABLE, variable, INTEGER *type, char **szData, double *nData). For objects (type is VARIABLE_ARRAY, VARIABLE_CLASS or VARIABLE_DELEGATE) in szData you will find the object reference. Just cast it to a (void *). For delegates in nData you will have the member identifier.

Chapter 7

Coding guidelines

Concept programming language does not define a coding standard. However, a few tips will provide a consistent look to the code. I encourage all Concept programmers to write based on this suggestions.

7.1 Indentation suggestions

I recommend the use of 4-spaces tab for each level of indentation. Each begin bracket should be placed on the same line as its "owner", rather that on a new line.

```
if (a == 1) {
    do_something();
    do_something_more();
}
```

For on-line statements, it's not recommended to use the brace (easier on the eyes).

```
if (a == 1)
    do_something();
```

Each begin brace ({) should be followed by a new line, and will generate a new level of indentation(tab/4 spaces), except when defining properties.

if, else, while, do and *for* will generate a new level of indentation when not using braces(when using just one statement, it should be indented by a tab). *if* and *else* should always be on the same level.

```
class first_level {
    property second_level_property { set x, get x }

    second_level() {
        if (third_level)
            echo "fourth level";
    }
}
```

When dealing with multiple *if..else* branches, I recommend that the branch *if* should be on the next line after the previous *else*, instead of the same line. In this case only, the previous else will not generate a new indentation level(the following if will be on the same level with the previous else).

```
if (a == 1)
    echo "A is one";
else
if (a == 2)
    echo "A is two";
else
if (a == 3)
    echo "A is three";
```

Each *try..catch* should have braces, regarding of the number of statements. *try* and *catch* should always be on the same level.

```
try {
    doSomething();
} catch (var exc) {
    echo "Exception!";
}
```

switch will be always followed by a begin brace on the same line, and all the following *case* and *default* on the same indentation level. Both *case* and *default* should be immediately followed by “:”, a new line, and a new indentation level.

```
switch (a) {
    case 1:
        echo "A is 1";
        break;
    case 2:
        echo "A is 2";
        break;
    default:
        echo "A is default";
}
```

Always write only one statement per line. When using *if* and *while* with multiple conditions, enclose each condition by parenthesis.

```
while ((a > 0) && (a < 100))
    doSomething();
```

Keep at least one line of space between each function, at least one blank line between member variables and properties, pure virtual functions and triggers. Use at least one blank line between properties and other kind of member.

```
class A {
    var _a;
    var _b;

    property a { get _a }
    property b { get _b }

    foo() {
        echo "foo";
    }

    foo2() {
        echo "foo";
    }
}
```

When defining multiple variables or parameters, add a space after each comma.

```
function foo(a, b, c=2, d) {
   var e, f, g;
}
```

When declaring arrays, add a space after each comma. When defining a matrix as an array of arrays, each matrix line should be on its own code line.

```
var matrix =  [[11, 12, 13, 14, 15],
        [21, 22, 23, 24, 25]];
```

When initializing relatively small key-values arrays is better to have each key on its own line.

```
[
   "Key1": 1,
   "Key2": 2,
   "Key3": 3
];
```

Is better to avoid comments on the same line with a statement. I would recommend the use of the line comment instead of the block comment ("//" instead of "/* ... */"). Keep your comments lexically correct. Comments should respect its current indentation level.

```
if (true) {
   // This is how your comment should
   // look like.
}
```

When possible, put spaces between the binary operators, except for selectors(. -¿ and array[index]). Unary operators should not be followed by any space. Operators and operands should be on the same line of code. Avoid breaking small statements into multiple lines of code.

```
var b = 2;
var a = 1 + b;
var n = 0;
if ((a > 2) && (b < 3) && (!n))
   echo "Here!";
```

7.2 Variable and member naming

It's recommended that object variables identify it's owner class.

```
// if just one duck used
var duck = new Duck();
```

or

```
var first_duck = new Duck();
var secondDuck = new Duck();
```

In the above example, I've used two naming conventions: one using an underscore, and the other one using a capital letter. Each of them are clear enough on the type of the variable.

Is better to reserve one letter variables for local loops. Simpler names are best used with number variables.

```
var index = 1;
var len = length "123";
```

For string and arrays, I recommend the use of the *str*, respectively *arr* prefix. When a function has only a few lines a code, you can skip this prefixes.

```
var arrNames = ["Eduard", "Mike"];
var strName = "Eduard";
```

You should use any naming convention you like, but use it in your entire project. Keep in mind that somebody will need to read your code at some point.

Classes must not be named used common variable names like “n” or “arr”. Class member must have clear representative names. Use full words for member names.

```
class Person {
   var Name = "";
   var Address = "";
```

```
    var BillingAdddress = "";
}
```

I like prefixing private and protected members with a “_”, in order to help Concept IDE code competition(will appear last in auto-complete lists), but you shouldn’t consider this a rule.

7.3 Circular references

When a variable is referencing an object, that references another object referencing the initial object you have a circular reference.

This is how a circular reference looks like:

```
var a = new A();
a.SomeArray["SomeKey"] = a;
```

Is better to avoid this, because it can generate a little overhead for the Garbage Collector when clean it. This is to be avoided from logical reasons, more than technical.

An object referencing itself is not really a circular reference, so this should be ok:

```
var a = new A();
a.SomeObject = a;
```

7.4 Love your code

Remember that coding should be fun. When it isn’t fun, it means that you either need a break, a new boss or a new job. Coding is not a business, is an art, and art has no deadlines. Real artists are driven by pure passion and desire to create, not by the highest bidder.

Your code will never be perfect, just good enough for production. Take a few hours once in a while, and write code for your own projects or fun. If you don't like coding, you've chosen the wrong line of work.

Part III

Concept Framework

Concept Application Server comes with a rich, fully cross-platform and OS-independent framework. It covers all the user-interface, I/O, including database abstraction layer.

It has similar features with well-known frameworks like GTK+ or .NET, for the programmer to get relatively fast used with it.

We will consider the framework as a set of two API's: static API's, implemented in C, for best speed, and the actual framework objects, implemented in Concept.

Except the UNIX sockets API's, the frameworks should behave exactly the same on Windows, Linux, BSD and Mac OS X. Mobile devices run a stripped down version of the framework, bat the basic properties should work exactly the same as on desktop computers.

We will discuss in the next chapters the principles behind the Concept Framework rather than focusing on every single object, method, event or property. For that you have the on-line help(and off-line) available.

You will notice that every UI class begins with "R". This stands from Remote, for example "RLabel" should be read "Remote Label". Every class that has an impact on the client server has this "R" or "Remote" prefix. These classes are of little to no use in http and console applications. Classes that don't affect the client have no prefix (for example "Xapian").

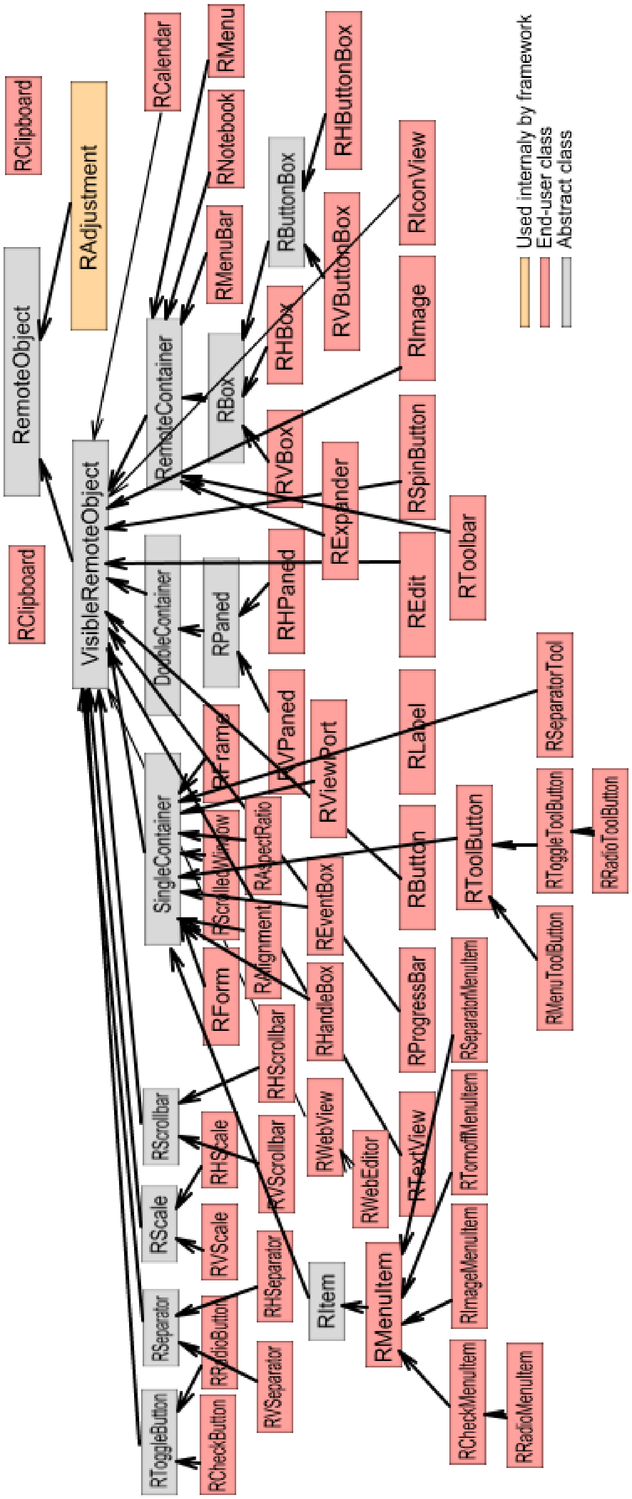


Figure 7.1:
The UI class hierarchy (basic classes)

Chapter 8

Cloud UI application architecture

8.1 Base objects

Every Concept remote UI object is a descendant of RemoteObject. RemoteObject keeps a remote identifier, which can be accessed by the RID property. All visible objects descend from VisibleRemoteObject, which hold basic object properties like *Visible, Enable, Tooltip, FgColor (foreground color), BgColor, Font, DragIcon, Packing, MinWidth, MinHeight*, and the read-only properties *Width* and *Height*.

Let's define a widget as an object that can't have any children. Every widget is usually a direct descendant of VisibleRemoteObject. Examples of widgets are RLabel, REdit.

A container is an object that can have children. Every container must inherit *RemoteContainer*, *SingleContainer* or *DoubleContainer*. *RemoteContainer* can have any number of children. Examples of *RemoteContainer* are *RVBox* and *RTable* which are classes used for layouts. *RemoteContainer* adds the *Childs* property, an array containing all its children.

Single containers are objects that can have only one child, accesible by using the *Child* property. Most of UI visible objects are single containers, for example: *RForm, RButton, RFrame* and *RExpander*. These are objects that hold just one child (any kind), usually, but not limited to, a *RemoteContainer*, for example *RTable*.

Double containers are used just by the RPaned, RVPaned and RHPaned, a kind a control that allows the user to resize the area occupied by left or top, right or bottom by using the mouse. *DoubleContainer* adds two properties: *Child1* and *Child2*. These controls are not available on mobile devices.

Every UI application must instantiate the *CApplication* class(short for Concept Application), defined in Application.con. This is how the minimal concept:// application should look like:

minimalApplication.con

```
include Application.con
include RForm.con

class Main {
   function Main() {
         try {
            var Application = new CApplication(new RForm(null));
            Application.Init();
            Application.Run();
            Application.Done();
         } catch (var Exception) {
            echo Exception;
         }
   }
}
```

The previous example simply creates an empty window. The *null* parameter used in RForm's constructor means that it has no parent.

CApplication's *Init*() method prepares the client for the execution, *Run*() starts executing the applications and lunches the message loop and the *Done*() method cleans up and notifies the client that the application ended correctly.

Note that this application doesn't produce any result if ran from a concept console. The correct way of running this application is to copy it in the

Concept Application Server root (refer to the first chapters), and run it by using Concept Client using the address:

```
concept://localhost/minimalApplication.con
```

or, if you use CIDE (Concept IDE), and place the application into MyProjects/minimalApplication/ folder:

```
concept://localhost/MyProjects/minimalApplication/minimalApplication.con
```

All remote UI objects take the owner object as the sole parameter for their constructor. Also, all VisibleRemoteObject-derived UI objects are not visible by default. You should call *Show*() or set the *Visible* property to true.

UI objects have a variety of events, for example *OnShow*. All the events, regarding object and event type have two parameters: *Sender*(object) and *EventData*(string). If an event is needed, you can set the event handler like this:

```
[...]
    MyForm(Owner) {
        super(Owner);
        // set the form title
        this.Caption = "Hello world!";
        button1 = new RButton(this);
        button1.OnShow = this.OnShow;
    }

    OnShow(Sender, EventData) {
        CApplication.Message("Shown!");
    }
[...]
```

Note that events set on an object without parent, except the main form, will not work. You need first to call LinkIfNotLinked(obj) where object is a reference to main form or any other UI object that has a parent. Objects without parent are those created with *null* as a parent, for example *new RImage(null)*.

Avoid doing extensive processing in events, because it will halt the main message loop, resulting in a delay of processing the other events.

Hiding the main form, causes the application execution to end.

An application must have at least one form. It can create any number of additional forms, if needed, just be sure to create the child windows with main form as the parent.

Some RForm members that you may use frequently:

Caption : string property
the title of the form

Resizable : boolean property (default is true)
allow the user to resize the form if true (on mobile clients has no effect)

Modal : boolean property (default is false)
if set to true, the use will not be able to interact with any other form except this (the modal form). It has no effect on mobile clients.

CSS : string property (inherited from VisibleRemoteObject)
set the CSS (Cascading Style Sheets) for the given form and/or its children.
For example, a gradient background may be set for a form and its child forms:

```
form.CSS = "RForm { background-color:
    qlineargradient(x1:0, y1:0, x2:0, y2:1, stop:0 #f1f1f1
    stop:1 #e4f0f5); }";
```

Maximized : boolean property (default is false)
if set to true, the form will be maximized. It has no effect on mobile clients.

Minimized : boolean property (default is false)
if set to true, the form will be minimized. It has no effect on mobile clients.

FullScreen : boolean property (default is false)
if set to true, the form will use the entire screen. It has no effect on mobile clients.

Decorations : property (DECOR_ALL, DECOR_BORDER, DECOR_RESIZEH, DECOR_TITLE, DECOR_MENU, DECOR_MINIMIZE, DECOR_MAXIMIZE)
sets the available decorations (buttons in title bar). It has no effect on mobile clients.

SkipTaskbar : boolean property (default is false)
if set to true, will be hidden in your OS task bar. It has no effect on mobile clients.

KeepAbove : boolean property (default is false)
if set to true, will be the window will appear on top of other windows. It has no effect on mobile clients.

KeepBelow : boolean property (default is false)
if set to true, will be the window will appear on beneath other windows. It has no effect on mobile clients.

Icon : RImage property
Sets an RImage that would be used as form's icon. It has no effect on mobile clients.

Position : property (WP_NONE, WP_CENTER, WP_MOUSE, WP_CENTER_ALWAYS, WP_CENTER_ON_PARENT)
Sets the window default position. It has no effect on mobile clients.

Closeable : boolean property
if set to true, the user will be able to close the window.

Opacity : number property
Sets the opacity of the window, with 0.0 being fully transparent(invisible) and 1.0 being completely opaque.

Screen : number property
Sets the screen/monitor index in which the window will be shown

Width : number property
Sets or gets the window width in pixels. It has no effect on mobile clients.

Height : number property
Sets or gets the window height in pixels. It has no effect on mobile clients.

Left : number property (inherited from VisibleRemoteObject)
Sets or gets the window X coordinate of the top left corner. It has no effect on mobile clients.

Top : number property (inherited from VisibleRemoteObject)
Sets or gets the window Y coordinate of the top left corner. It has no effect on mobile clients.

MouseCursor : number property (inherited from VisibleRemoteObject)
Sets or gets the mouse cursor (stock cursor). Some possible values are: CURSOR_CURSOR_ARROW, CURSOR_CLOCK, CURSOR_TOP_RIGHT_CORNER. See Concept Framework documentation for VisibleRemoteObject for a list with all the possible values (about 80 stock cursors). It has no effect on mobile clients.

MouseCursorImage : RImage property (inherited from VisibleRemoteObject)
Sets or gets the mouse cursor from the given RImage(an image of any size). It has no effect on mobile clients.

Restore ()
Restores the minimized or maximized window

Maximize ()
Maximizes the window

Minimize ()
Minimizes the window

BacktroundImage (RImage img)
Sets *img* as the background image

Show () inherited from VisibleRemoteObject
Shows the window. It is equivalent to RForm.Visible = true.

Hide () inherited from VisibleRemoteObject
Hides the window. It is equivalent to RForm.Visible = false.

Here are most used events in applications (associated with all widgets, not only RForms). All events must be handled by delegates with two parameters: EventHandler(RemoteObject Sender, string EventData).

OnRealize inherited from VisibleRemoteObject
Called when the widget is created on the client

OnShow inherited from VisibleRemoteObject
Called when the widget is shown

OnHide inherited from VisibleRemoteObject
Called when the widget is hidden

OnFocus inherited from VisibleRemoteObject
Called when the widget receives focus

OnFocusOut inherited from VisibleRemoteObject
Called when the widget looses focus

OnExposeEvent inherited from VisibleRemoteObject
Called when the widget become visible on the screen

OnKeyPress inherited from VisibleRemoteObject
Called when the user presses a key. EventData contains the key code.

OnKeyRelease inherited from VisibleRemoteObject
Called when the user releases a key. EventData contains the key code.

OnButtonPress inherited from VisibleRemoteObject
Called when the user presses a mouse button. EventData contains the mouse button index.

OnButtonRelease inherited from VisibleRemoteObject
Called when the user releases a mouse button. EventData contains the mouse button index.

OnDragBegin inherited from VisibleRemoteObject
Called when the user begins a drag and drop operation. Note that the Dragable property must be set to true and DragData to a string describing the drag operation (that will be sent to the drop site).

OnDragEnd inherited from VisibleRemoteObject
Called when the user finishes a drag and drop operation. Note that the Dragable property must be set to true and DragData to a string describing the drag operation (that will be sent to the drop site).

OnDragDataReceived inherited from VisibleRemoteObject

Called when the user is in the process of a dropping something on the UI object. Note that the *DropSite* property must be set to DROP_STRINGS, DROP_FILES or DROP_ANY (default is DROP_NONE). EventData contains the object's drag data. If DropSite is DROP_FILES you can get the dropped files via GetDNDFile(strinf filename, var content, var error_descripton = ""). If you want to cancel the drag and drop operation, you may use ResetDNDFiles().

```
public on_DragDataReceived(VisibleRemoteObject Sender, string
    EventData) {
    // file names are separated by "\r\n"
    var arr = StrSplit(EventData,"\r\n");
    var len = length arr;
    for (var i=0;i<len;i++) {
        // gets the full path of the filename
        var filename = URIDecode(arr[i]);
        if (!this.MediaView.GetDNDFile(filename, var content,
            var errtext)) {
            // filename is the client filename (including path)
            // content contains the file content.
            // TO DO: process the data
        } else
            echo "Error!";
    }
}
```

OnDragDrop inherited from VisibleRemoteObject

Called when the user drops data on the UI object. Note that the *DropSite* property must be set to DROP_STRINGS, DROP_FILES or DROP_ANY (default is DROP_NONE). EventData contains the path where the data was dropped for tree views, list views and icon views, and x and y coordinates for any other object, in the form "X:Y".

Some objects don't fire all the events, for example *RLabel* or *RImage*. If it's needed to handle a click or a key event for a static UI object (objects without interaction from the user), an *REventBox* must be added as the static object's parent, and handle the needed on the event box.

Instead of directly adding the label to its parent:

```
var label = new RLabel(owner)
label.Show();
```

an intermediate event box should be added:

```
var eventbox = new REventBox(owner);
eventbox.Show();

var label = new RLabel(eventbox)
label.Show();
```

For the full list of events and properties, please check the Concept Framework documentation.

8.2 Relative layouts

Concept Framework's UI objects use almost exclusively relative layouts. There is a *RFixed* surface, that allows you to use absolute positioning, but I don't recommend it (and is unsupported on mobile platforms).

Layouts are based on *RVBox, RHBox, RTable, RVPaned, RVButtonBox and RHButtonBox* and *RHPaned*. RVBox is short from remote vertical box and RHBox from horizontal box. Each UI object has a property called *Packing* which has three possible values: PACK_SHRINK, PACK_EXPAND_PADDING and PACK_EXPAND_WIDGET. On mobile platforms PACK_EXPAND_PADDING and PACK_EXPAND_WIDGET produce the same effect. *RVButtonBox* and *RHButtonBox* are subclasses of RBox, adding the Layout property (BUTTONBOX_SPREAD, BUTTONBOX_EDGE, BUTTONBOX_START, BUTTONBOX_END) controlling how the child widgets spread, the *XPadding* and *YPadding* controlling the horizontal, respectively the vertical padding in pixels.

When using PACK_SHRINK, the widget will use the minimum amount of space (just enough to render its contents). PACK_EXPAND_PADDING will render the object as PACK_SHRINK, but will use all the available space as padding (not actually using the space, but rather "reserving" it). PACK_EXPAND_WIDGET will enlarge the UI object to use the entire

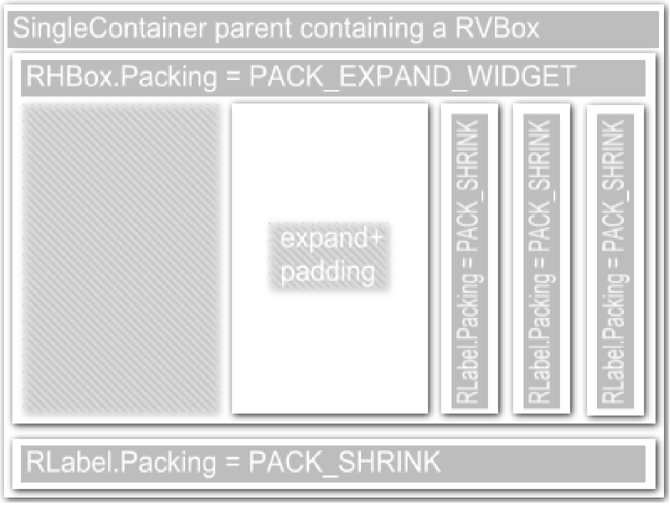

SingleContainer parent containing a RVBox
RHBox.Packing = PACK_EXPAND_WIDGET
expand+
padding
RLabel.Packing = PACK_SHRINK
RLabel.Packing = PACK_SHRINK
RLabel.Packing = PACK_SHRINK
RLabel.Packing = PACK_SHRINK

Figure 8.1:
Packing with vertical boxes

available space in its parent.

This packing system is identical with GTK+ packing (the framework on which the initial Concept Client was built). Even though not all Concept Clients are GTK+ based (like the Android or iOS versions), the behavior was implementing, because the flexible and screen resolution independent nature.

Boxes allow you to set the spacing between the children using the *Spacing* property (in points, as a number). Also, when the *Homogeneous* property is set to true, all of the boxes children use the same amount of space.

The tables are more flexible structures that allows you to create complex layouts. Each table has cells, and each child object can spawn over one or more cells, either vertically or horizontally. This can be controlled by calling the *AttachOptions* function before adding a new child. Not that a cell cannot contain more than one child (you cannot have overlapping objects in a table).

```
RTable.AttachOptions(left_cell, right_cell, top_cell, bottom_cell,
    horizontal_options=FILL_EXPAND, vertical_options=FILL_EXPAND,
    horizontal_spacing=0, vertical_spacing=0)
```

The vertical and horizontal options takes one of the following constants as a parameters:

EXPAND
expands the child object

SHRINK
shrinks the child object to its minimum size

FILL
fills the available space (used as a mask for EXPAND and SHRINK)

FILL_EXPAND
alias for EXPAND | FILL

The cell spacing can be controlled using the *RowSpacing* and *ColSpacing* properties. There is also a *Homogeneous* property that makes cells equal when set to true.

By using boxes and tables you can create any kind of layouts, without being pixel-dependent.

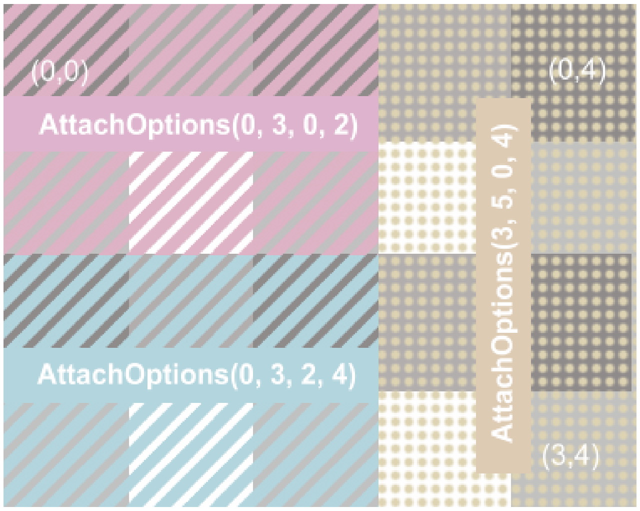

(0,0)
AttachOptions(0, 3, 0, 2)
AttachOptions(0, 3, 2, 4)
(0,4)
AttachOptions(3, 5, 0, 4)
(3,4)

Figure 8.2:
Table layout

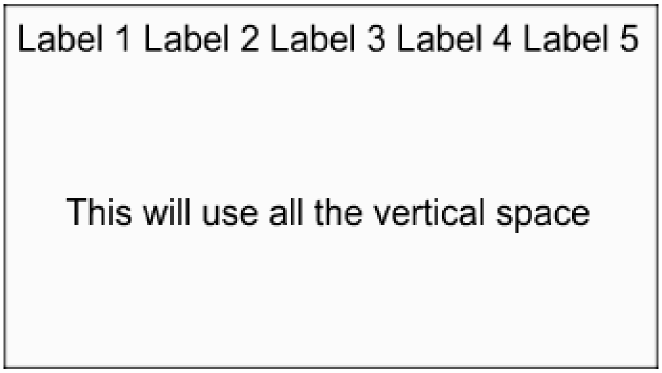
Every Concept UI application must have at leas one form. We will extend the basic *RForm* and will create a new form illustrating the relative layout behavior.

boxes.con

```
include Application.con
include RForm.con
include RHBox.con
include RVBox.con
include RLabel.con

class MyForm extends RForm {
   MyForm(Owner) {
      // call RForm's constructor
      super(Owner);

      var vbox=new RVBox(this);
      vbox.Show();

      var hbox=new RHBox(vbox);
      hbox.Packing=PACK_SHRINK;
      hbox.Show();

      for (var i=1;i<=5;i++) {
         var tmp_label = new RLabel(hbox);
         tmp_label.Caption = "Label $i ";
         tmp_label.Packing = PACK_SHRINK;
         tmp_label.Show();
      }

      var label1 = new RLabel(vbox);
      label1.Packing = PACK_EXPAND_WIDGET;
      label1.Caption = "This will use all the vertical space";
      label1.Show();
   }
}

class Main {
   Main() {
         try {
            var Application = new CApplication(new MyForm(null));
            Application.Init();
            Application.Run();
            Application.Done();
            } catch (var Exception) {
                echo Exception;
            }
    }
}
```

The output looks like figure 8.3. You can combine vertical boxes, horizontal boxes and tables anyway you please.

You will notice that relative layouts are very responsive to window size/resize.



This will use all the vertical space

Figure 8.3:
boxes.con output

A similar application can be created using *RTable* instead of *RVBox* and *RHBox*. I personally like tables because of the tidy look given to the form.

tables.con

```
include Application.con
include RForm.con
include RTable.con
include RLabel.con
include REdit.con
include RTextView.con

class MyForm extends RForm {
    MyForm(Owner) {
        super(Owner);
        var table = new RTable(this);
        table.Rows = 3;
        table.Columns = 2;
        table.Show();

        // left, right, top, bottom,
        // shrink horizontally, shrink vertically
        table.AttachOptions(0, 1, 0, 1, SHRINK, SHRINK);
        var labelName = new RLabel(table);
        labelName.Caption = "Your name";
        labelName.Show();

        // expand horizontally filling the empty
        // space, shrink vertically
        table.AttachOptions(1, 2, 0, 1, EXPAND | FILL, SHRINK);
        var editName = new REdit(table);
        editName.Show();

        table.AttachOptions(0, 2, 1, 2, SHRINK, SHRINK);
        var labelAddress = new RLabel(table);
        labelAddress.Caption = "Address";
        labelAddress.Show();

        // expand horizontally, expand vertically
        table.AttachOptions(0, 2, 2, 3, EXPAND | FILL, EXPAND |
            FILL);
        var viewAddress = new RTextView(table);
        viewAddress.Show();
    }
}

class Main {
    Main() {
            try {
                var Application = new CApplication(new MyForm(null));
                Application.Init();
                Application.Run();
                Application.Done();
            } catch (var Exception) {
                echo Exception;
            }
    }
}
```

The output looks like figure 8.4. By combining tables with boxes you can easily get any layout you need, remaining pixel-independent.

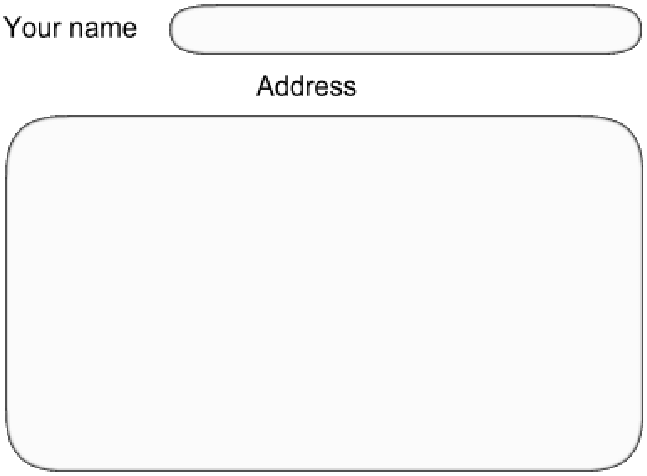


Figure 8.4:
tables.con output

Note that you can use *RScrolledWindow* to add scrolling capability to your objects.

For example:

```
[..]
var scrollWindow = new RScrolledWindow(this);
scrollWindow.Show();

var table = new RTable(scrollWindow);
table.Show()

[..]
```

,

This will make your table scrollable by the user, if its content don't fit in the given area. *RScrolledWindow* uses *HScrollPolicy* and *VScrollPolicy* to control when the scroll bars will be shown(POLICY_ALWAYS(default), POLICY_AUTOMATIC(preferred), POLICY_NEVER).

8.3 MDI with notebooks

When dealing with multiple documents, it's cleaner and easier to keep the user's attention if you use a *RNotebook* instead of multiple windows.

Concept notebooks(known also as tab-bars) provide an easily customizable interface. It behaves like a box showing just one child at a time. A child is rendered on a notebook page, and each page has a caption that can be a text, image(supported on all platforms), or any other kind of object, a button for example this being supported only on desktop computers. You can set/get the current page index by using the *PageIndex* property. On desktop computer you can control the tabs visibility by setting *ShowTabs*(true or false) and the position by setting *TabPos* to POS_LEFT, POS_RIGHT, POS_TOP(default) or POS_BOTTOM.

Note that on Android tabs are show only at the top and on iOS only at the bottom of the display and cannot be hidden.

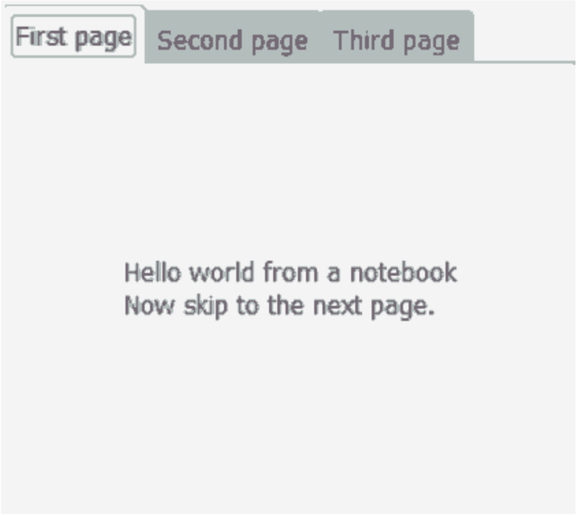
Each notebook page can contain only one UI object(for example a *RVBox* or *RTable*). There is no limit on the number of pages, though on iOS recommends a maximum of 5 tabs(when more than 5, a more button appears with the other pages). RNotebook implements the *OnSwitchPage(RNotebook Sender, string EventData)* event which notifies you when a user changed a page. It is not mandatory to catch this event. In the EventData parameter the new page index is received, in order to avoid calling PageIndex and generate a delay by a new exchange of messages on the network.

The notebooks captions can be set via de Pages property (array of objects). For example, after adding the first object to a RNotebook, we can call *notebook.Pages[0].Caption="First page"* or *notebook.Pages[0].Header=new RImage(null)* if we want to use an image instead of an actual text.

notebook.con

```
include Application.con
include RNotebook.con
include RLabel.con

class MyForm extends RForm {
   MyForm(Owner) {
       super(Owner);

       var notebook = new RNotebook(this);
       notebook.Show();

       var label1 = new RLabel(notebook);
       label1.Caption = "Hello world from a notebook\nNow skip to
           the next page.";
       label1.Show();
       notebook.Pages[0].Caption = "First page";

       var label2 = new RLabel(notebook);
       label2.Caption = "Hello again!";
       label2.Show();
       notebook.Pages[1].Caption = "Second page";

       var label3 = new RLabel(notebook);
       label3.Caption = "Please go to the first page";
       label3.Show();
       notebook.Pages[2].Caption = "Third page";

       // we could show the 3rd page (first page has index 0)
       // notebook.PageIndex = 2;
   }
}

class Main {
   Main() {
           try {
               var Application = new CApplication(new MyForm(null));
               Application.Init();
               Application.Run();
               Application.Done();
           } catch (var Exception) {
               echo Exception;
           }
   }
}
```

The output looks like figure 8.5. You can use any form of layout like *RTable* or *RVBox* instead of the *RLabel*(used in the previous example).



Hello world from a notebook
Now skip to the next page.

Figure 8.5:
notebook.con output

8.4 Text fields

Concept Frameworks defines three types of text fields:

RLabel
static text, not editable by the user

REdit
Signle-line edit box

RTextView
Multiple rich text edit box

Each of these can use custom fonts by using Font property(see *RFont* class reference in Concept Documentation).

RLabel uses two alignment properties: *XAlign*(supported also by *REdit*) and *YAlign*. These properties control how the text is aligned inside the control. They must have a real value between 0.0 and 1.0. For XAlign, 0.0 means left-aligned, 0.5 center and 1.0 right aligned. Similar, for YAlign, 0.0 means at aligned at top, 0.5 vertical center, and 1.0 at the bottom.

For RLabel you can set the *Caption* property and for REdit and RTextView, the *Text* property sets or gets the text.

REdit and RTextView can be used as static text if you set by setting the *ReadOnly* property to true.

RLabel supports some HTML tags like <*b*>(bold), <*i*>(italic), <*font*> and <*span*>. On mobile devices you can use any HTML tags you need, but on desktop, you are limited to Pango markup language. If you want to use tags, you need to set the *UseMarkup* property to true first.

REdit supports password mode by setting the *Masked* property to true. When in password mode, the user will see *(masked char) instead of the typed characters.

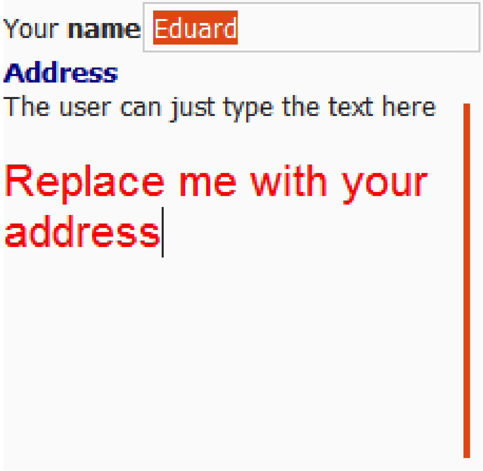
All of these controls must receive UTF-8 strings. UTF-8 is the only format used by the Concept Framework (UI). If you need to output text from other formats, use the *iconv2* function (see Concept Framework help) to convert it to UTF-8.

When using RTextView you can define multiple styles of text, by using the *CreateStyle(string stylename)* function. This allows you to create RTextTag objects with which you can manipulate specific parts of the text. Then you can add custom text by using *AddStyledText(string text, RTextTag style)*. You can also add objects like buttons or images using the *AddObject(object)* method. The *Wrap* property can be set to WRAP_NONE(default), WRAP_CHAR, WRAP_WORD, WRAP_WORD_CHAR to control how the text is wrapped.

textfields.con

```
include Application.con
include RTable.con
include RLabel.con
include REdit.con
include RTextView.con
include RScrolledWindow.con

class MyForm extends RForm {
    MyForm(Owner) {
        super(Owner);
        var table = new RTable(this);
        table.Rows = 3;
        table.Columns = 2;
        table.Show();

        // left, right, top, bottom,
        // shrink horizontally, shrink vertically
        table.AttachOptions(0, 1, 0, 1, SHRINK, SHRINK);
        var labelName = new RLabel(table);
        labelName.UseMarkup = true;
        labelName.Caption = "Your <b>name</b>";
        // align left
        labelName.XAlign = 0.0;
        // align top
        labelName.YAlign = 0.0;
        labelName.Show();

        // expand horizontally filling the empty
        // space, shrink vertically
        table.AttachOptions(1, 2, 0, 1, EXPAND | FILL, SHRINK);
        var editName = new REdit(table);
        editName.Text = "Eduard";
        editName.Show();

        table.AttachOptions(0, 2, 1, 2, EXPAND | FILL, SHRINK);
        var labelAddress = new RLabel(table);
        labelAddress.Caption = "Address";
        labelAddress.Font.Name = "bold";
        // set label color to dark blue (0x R G B in hexadecimal)
        labelAddress.FgColor = 0x000080;
        labelAddress.XAlign = 0.0;
        labelAddress.YAlign = 0.0;
        labelAddress.Show();

        // expand horizontally, expand vertically
        table.AttachOptions(0, 2, 2, 3, EXPAND | FILL, EXPAND |
            FILL);
        // add scrollbars to textview
        var scrollAddress = new RScrolledWindow(table);
        scrollAddress.HScrollPolicy = POLICY_AUTOMATIC;
        scrollAddress.Show();

        var viewAddress = new RTextView(scrollAddress);
        viewAddress.Text = "The user can just type the text
            here\n\n";
        viewAddress.Wrap = WRAP_WORD;
        viewAddress.Show();

        var style=viewAddress.CreateStyle("My Style 1");
        style.FontName = "Arial 16";
        style.ForeColor = 0xFF0000;
        viewAddress.AddStyledText("Replace me with your address",
            style);
    }
}

class Main {
    Main() {
        try {
            var Application = new CApplication(new MyForm(null));
            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            echo Exception;
        }
    }
}
```

Figure 8.6 shows the output.



Your name Eduard

Address

The user can just type the text here

Replace me with your address

Figure 8.6:
textfields.con output

For *REdit* you have the *Suggest* string property that allows the user to select a preset value from a given set, while typing. The suggested elements must be separated by ";", for example: *editName.Suggest = "Ana;Claudia;John;Maria"*. Alternatively you can use the *SuggestModel(object, text_column_index)* function to populate the suggestion list from an existing tree view, icon view, combo box or editable combo box.

For each *REdit* you can set two icons, primary(at edit field's start) and secondary(at edit field's end) via string property *PrimaryIconName*(stock icon name) or *PrimaryIcon* (RImage), respectively *SecondaryIconName* and *SecondaryIcon*. If *PrimaryIconActivable* and/or *SecondaryIconActivable* is set to true, you can handle the *OnIconPress* or *OnIconRelease* events to be notified when an user clicked on the icons. The EventData parameter will be "0" for the primary icon or "1"(as string) for the seconday icon. For example, you can set edit.SecondaryIconName = "gtk-find" for showing the search icon at the end of the field.

8.5 Buttons

You can use multiple type of buttons, starting with push buttons(RButton), check buttons(RCheckButton), radio buttons(RRadioButton) and multiple type of tool buttons.

RButton is a single container that can hold text, images or both. When clicked it fires the OnClicked event, if handled. Being a single container, allows you to add anything you like inside the button. Usually just text and images, but you could add a RLabel in order to use markup text.

RCheckButton is a toggle button(VisibleRemoteObject) that can be checked, unchecked or in a inconsistent mode (nor checked nor unchecked). You can access the state of the button by using the *Checked* property (true or false) or set it to an inconsistent mode by setting the *Inconsistent* property to true.

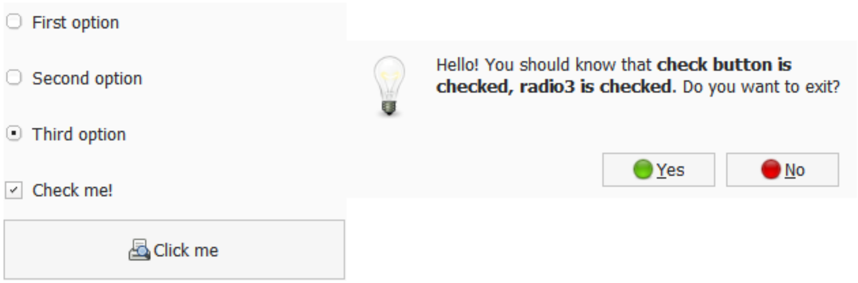
RRadioButton is also a toggle button, similar with *RCheckButton* but grouped with *GroupWith* property. When a radio button in a group is checked, the previous checked button is automatically unchecked. In a group, just one radio button can be checked at a time. Note that on mobile platforms the *GroupWith* property may be ignored on some platforms. Instead, the radio buttons will be automatically grouped by having the same parent. However, you should always set the *GroupWith* property, regardless of the target platform.

For all button type, you can set the displayed text by setting the *Caption* property(string).

buttons.con

```
include Application.con
include RVBox.con
include RButton.con
include RCheckButton.con
include RRadioButton.con
include StockConstants.con

class MyForm extends RForm {
    protected var radio1;
    protected var radio2;
    protected var radio3;
    protected var check;

    MyForm(Owner) {
        super(Owner);
        var box = new RVBox(this);
        box.Show();

        radio1 = new RRadioButton(box);
        radio1.Caption = "First option";
        radio1.Show();

        radio2 = new RRadioButton(box);
        radio2.Caption = "Second option";
        // group with radio 1
        // note that we don't need to set the GroupWith property
        // for button1
        radio2.GroupWith = radio1;
        radio2.Show();

        radio3 = new RRadioButton(box);
        radio3.Caption = "Third option";
        // group it with button1. We could also set it to radio2
        // because is in the same group with button1.
        radio3.GroupWith = radio1;
        radio3.Show();

        // check the first button
        radio1.Checked = true;

        // check is a member variable
        check = new RCheckButton(box);
        check.Caption = "Check me!";
        check.Show();

        var button = new RButton(box);
        button.Caption = "Click me";
        button.LoadStockIcon(PRINT_PREVIEW);
        button.OnClicked = this.ButtonClicked;
        button.Show();
    }

    ButtonClicked(Sender, EventData) {
        var result="";
        if (check.Checked)
            result+="check button is checked";
        else
            result+="check button is unchecked";

        if (radio1.Checked)
            result+=", radio1 is checked";
        else
        if (radio2.Checked)
            result+=", radio2 is checked";
        else
            result+=", radio3 is checked";
        // CApplication.Message uses markup text
        if (CApplication.Message("Hello! You should know that
            <b>$result</b>. Do you want to exit?", "Hello title",
            MESSAGE_INFO, BUTTONS_YES_NO) == RESPONSE_YES) {
            // hiding main form causes the application to end
            this.Hide();
        }
    }
}

class Main {
    Main() {
            try {
                var Application = new CApplication(new MyForm(null));
                Application.Init();
                Application.Run();
                Application.Done();
            } catch (var Exception) {
                echo Exception;
            }
    }
}
```

Figure 8.7 shows the output.



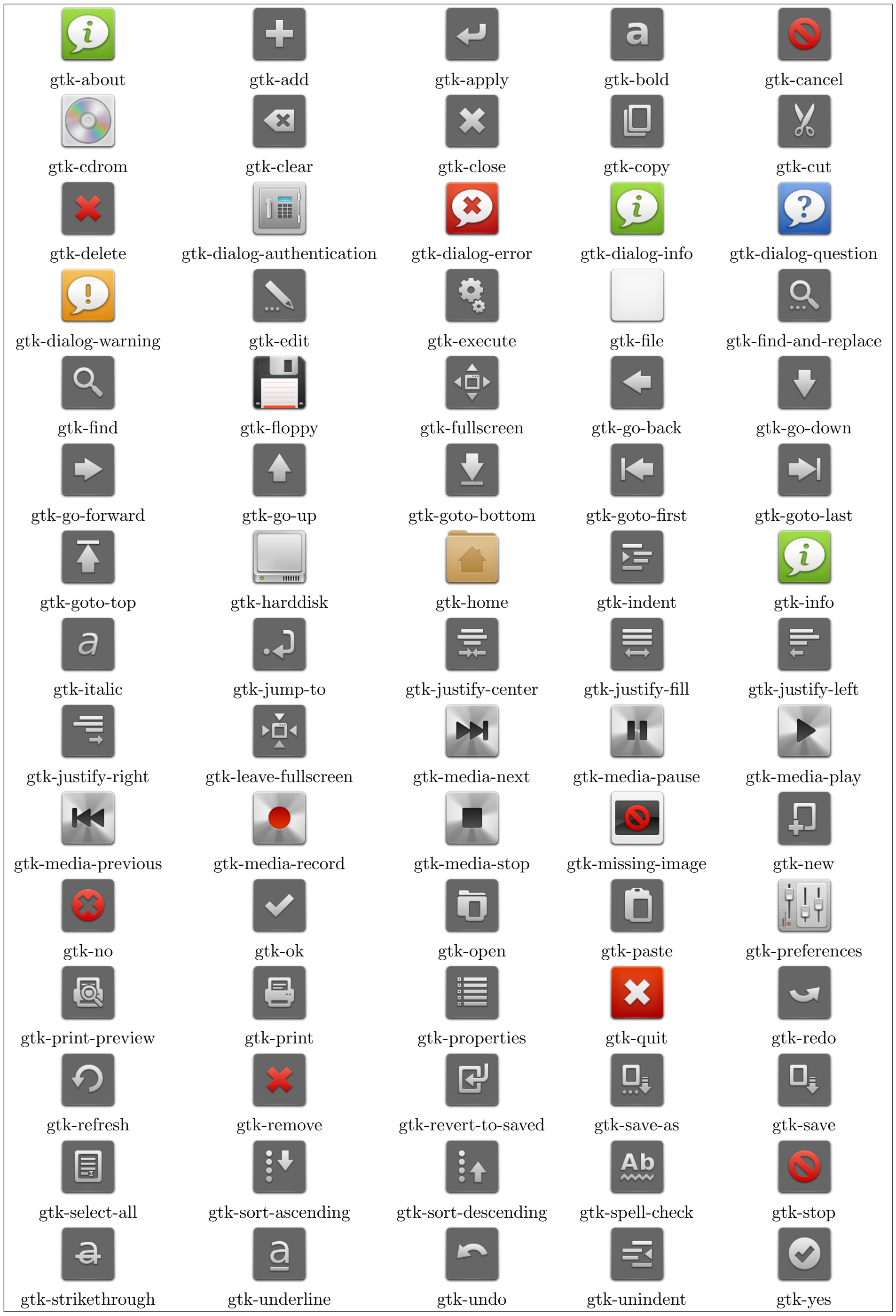
Hello! You should know that **check button is checked, radio3 is checked**. Do you want to exit?

Figure 8.7:
buttons.con output

8.6 Images

Images are managed by the *RImage* class. An *RImage* cannot have any children (being a direct VisibleRemoteObject descendant). An image object can use either a given image file or a stock image. Stock images are embedded in concept client (all platforms), that allows you to minimize the network traffic, and provide consistent look for Concept applications (for example, the “gtk-save” icon image will look as the native “Save as” icon on the given platform).

The stock images are prefixed by “gtk-”. The initial stock images were based on GTK+, but the list was extended to include more, and the “gtk-” prefix was maintained for consistency. The list of stock images contains:


gtk-about
gtk-add
gtk-apply
gtk-bold
gtk-cancel
gtk-cdrom
gtk-clear
gtk-close
gtk-copy
gtk-cut
gtk-delete
gtk-dialog-authentication
gtk-dialog-error
gtk-dialog-info
gtk-dialog-question
gtk-dialog-warning
gtk-edit
gtk-execute
gtk-file
gtk-find-and-replace
gtk-find
gtk-floppy
gtk-fullscreen
gtk-go-back
gtk-go-down
gtk-go-forward
gtk-go-up
gtk-goto-bottom
gtk-goto-first
gtk-goto-last
gtk-goto-top
gtk-harddisk
gtk-home
gtk-indent
gtk-info
gtk-italic
gtk-jump-to
gtk-justify-center
gtk-justify-fill
gtk-justify-left
gtk-justify-right
gtk-leave-fullscreen
gtk-media-next
gtk-media-pause
gtk-media-play
gtk-media-previous
gtk-media-record
gtk-media-stop
gtk-missing-image
gtk-new
gtk-no
gtk-ok
gtk-open
gtk-paste
gtk-preferences
gtk-print-preview
gtk-print
gtk-properties
gtk-quit
gtk-redo
gtk-refresh
gtk-remove
gtk-revert-to-saved
gtk-save-as
gtk-save
gtk-select-all
gtk-sort-ascending
gtk-sort-descending
gtk-spell-check
gtk-stop
gtk-strikethrough
gtk-underline
gtk-undo
gtk-unindent
gtk-yes

You can load an image, either by setting the *Filename* property, either by using the *LoadResource(string stock_constant, number size)* method, where size is 0, 1, 2, 3, 4 (0 meaning smallest icon, and 4 biggest).

images.con

```
include Application.con
include RVBox.con
include RImage.con

class MyForm extends RForm {
    MyForm(Owner) {
        super(Owner);
        var box = new RVBox(this);
        box.Show();

        var image = new RImage(box);
        // the same directory with the application
        image.Filename = "conceptclienticon.png";
        image.Show();

        var imageStock = new RImage(box);
        imageStock.LoadResource("gtk-print", 3);
        imageStock.Show();
    }
}

class Main {
    Main() {
            try {
                var Application = new CApplication(new MyForm(null));
                Application.Init();
                Application.Run();
                Application.Done();
            } catch (var Exception) {
                echo Exception;
            }
    }
}
```

Figure 8.8 shows the output.

On desktop computers you can use modifier functions like *Scale(width, height)*, *Rotate90(angle)* or *Flip(horizontal=false)* on the image object.

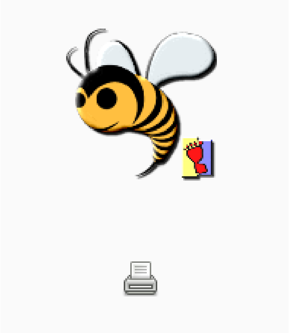


Figure 8.8:
images.con output

8.7 Menus and tool bars

In most applications you need methods of providing a menu to the user. This can be achieved using menu bars, pop-up menus or tool bars. All of these follow the same pattern.

Menu bars and pop-ups use the following UI objects:

RMenuBar
the menu bar as seen by the user. Can contain menu items and menus

RMenuItem
the standard menu item(as text)

RImageMenuItem inherits *RMenuItem*
a menu item with an attached image

RCheckMenuItem inherits *RMenuItem*
a menu item with behavior similar with a check button

RRadioMenuItem inherits *RMenuItem*
a menu item with behavior similar with a radio button

RTearoffMenuItem inherits *RMenuItem*
a menu item that can be detached from its parent by the user

RSeparatorMenuItem

a separator bar (with no interaction from the user)

An RMenu can be associated with the *PopupMenu* property available in all *VisibleRemoteObject* descendant objects. This will show the menu when the user right clicks on the given object. Same *RMenu* can be linked with multiple objects.

Note that static objects like RImage or RLabel don't catch the mouse events. If you want to attach a pop-up menu to one of these, you must put them in an *REventBox* object, and set the *PopupMenu* property of that event box. This is also useful when you want to detect mouse clicks in an image, handling the click by *OnButtonPress* or *OnButtonRelease*.

You can also add any type of object to a menu (for example an REdit) by using the *ContainedObject* property. This will allow you to replace a menu item with anything you like. For example *menuitem.ContainedObject = new REdit(null)*.

Menu items can have associated shortcut keys by setting the *AccelKey* string property. For example, if you want a menu item to be activated when the user presses Ctrl+s (Save), you could set *AccelKey* to "<control>s". Alternatively you can use the underscore("_") to prefix the menu shortcut key, for example menuitem.Caption="_File". In this case, Alt+F/Cmd+F will activate the menu.

RMenuItem fires the *OnActivate* event(if handled) when an user selects it by touching it or clicking it.

Alternatively you can use tool bars, using the same concept:

RToolbar
the tool bar as seen by the user. Can contain *RToolButton* descendant objects

RToolButton
the tool bar button (can contain text and/or image)

RToggleToolButton inherits *RToolButton*
a toggle button (similar with a check button)

RRadioToolButton inherits *RToggleToolButton*
a radio tool button

RToolSeparator
a separator b ar (with no interaction from the user)

Although they look and behave different, the logic behind them is nearly identical. In practice, you will create these objects visually by using Concept IDE's design view. To better understand how it works, we will create these objects from code.

Both menu bars and tool bars have limited support on mobile devices, due to lack of mouse and right click. Both on iOS and Android you could have just on menu bar per window, and tool buttons behave like typical buttons.

RToolButton fires the *OnClicked* event(if handled) when an user selects it by touching it or clicking it.

menus.con

```
include Application.con
include RVBox.con
include RImage.con
include RMenuBar.con
include REventBox.con

class MyForm extends RForm {
   MyForm(Owner) {
      super(Owner);
      var box = new RVBox(this);
      box.Show();

      var menubar = new RMenuBar(box);
      menubar.Show();

      var menuitem = new RMenuItem(menubar, "_File");
      menuitem.Show();
      // a menu to hold its children
      var menu = new RMenu(menuitem);
      menu.Show();

      // owner, caption, is_stock (default is false)
      var menuitem_new=new RImageMenuItem(menu, "gtk-new", true);
      menuitem_new.AccelKey="<control>n";
      // handle on activate (the user clicked the menu item)
      menuitem_new.OnActivate = this.NewActivate;
      menuitem_new.Show();

      var menuitem_save = new RImageMenuItem(menu, "Save");
      menuitem_save.AccelKey = "F2";
      menuitem_save.Show();
      var menuimage = new RImage(null);
      menuimage.Filename = "conceptclienticon.png";
      // scale image to 24x24
      menuimage.Scale(24, 24);
      // note that menuimage must not belong to another object!
      menuitem_save.SetImage(menuimage);

      var submenu = new RMenu(menuitem_save);
      submenu.Show();

      var radioMenu1=new RRadioMenuItem(submenu, "First file");
      radioMenu1.Checked = true;
      radioMenu1.Show();

      var radioMenu2=new RRadioMenuItem(submenu, "Second file");
      radioMenu2.GroupWith = radioMenu1;
      radioMenu2.Checked = false;
      radioMenu2.Show();

      var checkMenu=new RCheckMenuItem(submenu, "Check me out");
      checkMenu.Show();

      var menuitemEdit = new RImageMenuItem(menubar, "gtk-edit",
          true);
      menuitemEdit.Show();


      // use the menu also as a popup menu for image
      // we add an event box first, to capture the event
      // RImage and RLabel are static data, with no associated
      // window. When we want to catch a mouse event for these
      // objects, we need to put them in an REventBox.
      var eventbox = new REventBox(box);
      eventbox.PopupMenu = menu;
      eventbox.Show();

      var image = new RImage(eventbox);
      image.LoadResource("gtk-print", 3);
      image.Show();
```

```
    }

    NewActivate(Sender, EventData) {
        CApplication.Message("New!");
    }
}

class Main {
    Main() {
            try {
                var Application = new CApplication(new MyForm(null));
                Application.Init();
                Application.Run();
                Application.Done();
            } catch (var Exception) {
                echo Exception;
            }
    }
}
```

Don't be alarm by the line count. This code will be automatically generated for you by Concept IDE's design view.

Figure 8.9 shows the output.

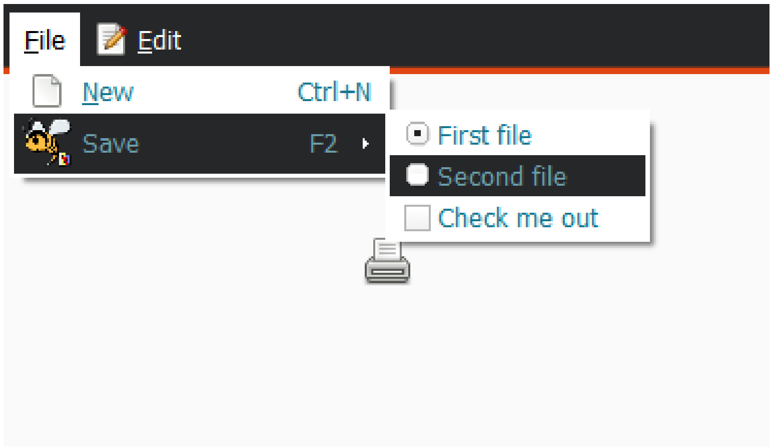


Figure 8.9:
menus.con output

For tool bars you can control the orientation by setting the *Orientation* property to ORIENTATION_HORIZONTAL(default) or ORIENTATION_VERTICAL. You can also control the tool bar style via the *Style* property and the following values:

TOOLBAR_ICONS
show only button icons

TOOLBAR_TEXT
show only button text

TOOLBAR_BOTH
show icon above and text below

TOOLBAR_BOTH_HORIZ
show icon at left and text at right

Now let's see the same principle applied to tool bars.

toolbars.con

```
include Application.con
include RVBox.con
include RImage.con
include RToolbar.con
include RToolButton.con
include RMenuToolButton.con
include RToolSeparator.con
include RMenu.con

class MyForm extends RForm {
   MyForm(Owner) {
       super(Owner);
       var box = new RVBox(this);
       box.Show();

       var toolbar = new RToolbar(box);
       toolbar.Show();

       var button1 = new RToolButton(toolbar);
       button1.Caption = "New";
       button1.StockID = "gtk-new";
       button1.Show();
       button1.OnClicked = this.OnButton1Clicked;
```

```
        var beeimage = new RImage(null);
        beeimage.Filename = "conceptclienticon.png";
        // scale image to 32x32
        beeimage.Scale(32, 32);
        beeimage.Show();

        var button2 = new RToolButton(toolbar);
        button2.Caption = "Bee";
        button2.Icon = beeimage;
        button2.Show();

        var separator = new RToolSeparator(toolbar);
        separator.Show();

        var menubutton = new RMenuToolButton(toolbar);
        menubutton.Caption = "Play";
        menubutton.StockID = "gtk-media-play";
        menubutton.Show();

        var menu = new RMenu(menubutton);
        menu.Show();

        var menuitem = new RMenuItem(menu, "Play a song");
        menuitem.Show();

        menubutton.Menu = menu;

        var image = new RImage(box);
        image.LoadResource("gtk-print", 3);
        image.Show();

    }

    OnButton1Clicked(Sender, EventData) {
        CApplication.Message("New!");
    }
}

class Main {
    Main() {
            try {
                var Application = new CApplication(new MyForm(null));
                Application.Init();
                Application.Run();
                Application.Done();
            } catch (var Exception) {
                echo Exception;
            }
    }
}
```

Figure 8.10 shows the output.

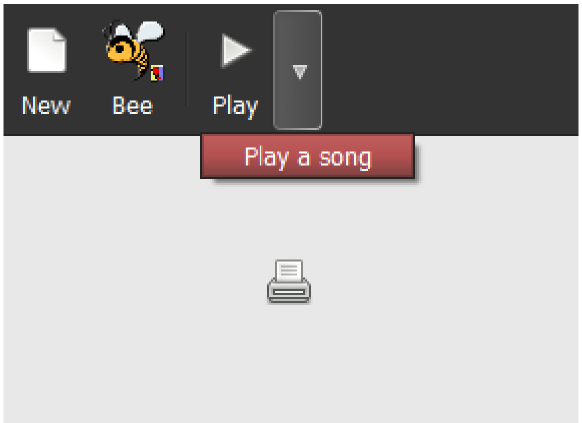


Figure 8.10:
toolbars.con output

8.8 Tree views, list views and combo boxes

Concept uses 4 types of element-based views:

RTreeView
a flexible object (VisibleRemoteObject) that allows you to display images, text, markup, progress bars, check and radio buttons in a multiple-column list or tree structure, with support for cell editing and event handling

RIconView
an object similar with RTreeView, in behavior, but displays only one icon and one text column.

RComboBox inherits *RToolButton*

a combo box, with one ore more columns (supporting all of the *RTreeView*'s column types)

REditComboBox a mixed object that has similar behavior both with *RComboBox* and *REdit*.

For each of these, the valid column types are:

HIDDEN_COLUMN
an invisible column

NORMAL_COLUMN
renders cells as text

PERCENT_COLUMN
renders cells as progress bars

CHECK_COLUMN
renders cells as check buttons

RADIO_COLUMN
renders cells as radio buttons

IMAGE_COLUMN
renders cells as images

COMBO_COLUMN
renders cells as combo boxes

MARKUP_COLUMN
renders cells as markup text (not usable for *RIconView* - alternatively you can use RIconView.MarkupColumn = column index)

EDITABLE_COLUMN
flag to be used with other column type, making the cell editable

EDITABLE_COLUMN must be combined with other column types, for example NORMAL_COLUMN | EDITABLE_COLUMN.

For *RTreeView* only, is mandatory to set the data model by using the *Model* property. Acceptable values are *MODEL_LISTVIEW* and

MODEL_TREEVIEW. You cannot change the model once columns are defined, just be sure to set the *Model* property before you do anything else with the view. When in tree view mode, you can add children to an item via *AddChild* or *InsertChild members.* This cannot be done for list views. Note that MODEL_TREEVIEW is to be avoided when writing applications for mobile platforms, due to its limited support.

RTreeView fires the *OnRowActivated* event, if handled, when user selects an item. The EventData string contains the selected row index or path (for tree views) in order to avoid additional network traffic by reading *RowIndex* property (which holds the same value). You can set *MultipleSelection* to true if you want to allow the user to select multiple rows. You can get or set the selected item by using the *RowIndex* property. If *MultipleSelect* is true, you can use the *Selected* property (read-only), and you will get an array containing the selected indexes. Note that indexes are returned as number for MODEL_LISTVIEW and as string for MODEL_TREEVIEW. The *Items* property allows you to access the elements in a tree view, icon view and combo box.

Items for tree views, icon views and combo boxes must be arrays with element count at least equal with the number of columns in the view. If more items are sent, than the number of columns, the elements after column count will be ignored.

You can manage the items by using the following functions:

AddItem (array item)
: Applies to RTreeView, RIconView, RComboBox, REditComboBox

InsertItem (position, array item)
: Applies to RTreeView, RIconView, RComboBox, REditComboBox

DeleteItem (position)
: Applies to RTreeView, RIconView, RComboBox, REditComboBox

UpdateItem (position, array new_item)
: Applies to RTreeView, RIconView, RComboBox, REditComboBox

AddChild (path_after, array item)
: Applies to RTreeView when model is MODEL_TREEVIEW

InsertChild (path_after, number index, array item)

Applies only to RTreeView when model is MODEL_TREEVIEW

Clear ()
: Clears all the items. Applies to RTreeView, RIconView, RComboBox, REditComboBox

ClearColumns ()
: Removes all the columns. Applies to RTreeView and RIconView.

ClearItemColumns ()
: Removes all the columns. Applies to RComboBox and REditComboBox

The position parameter can be a string, a number or an array(only for RTreeViews/MODEL_TREEVIEW). For list views is the index, for tree views can be either the index, or a string representing the path, for example "0:1:1:3" meaning Item 0, Child 1, Sub-child 1, sub-sub-child 3. You may use the array version instead of a string: [0, 1, 1, 3] being exactly the same as "0:1:1:3". If no item is selected, RowIndex will be -1.

RTreeView allows you to set *SearchColumn* property to, enabling the user to search elements in the view by simply typing on the keyboard

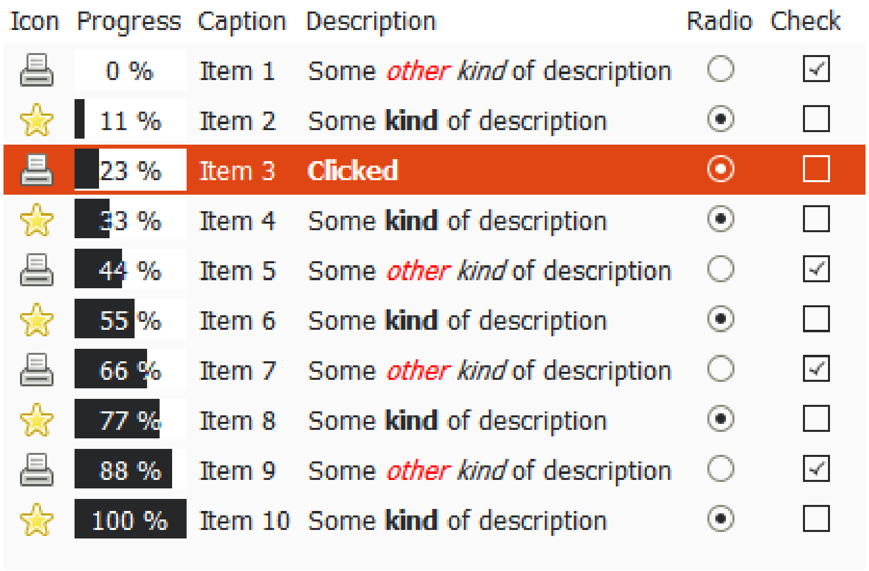
listviews.con

```
include Application.con
include RForm.con
include RTreeView.con
include RScrolledWindow.con
include RImage.con

class MyForm extends RForm {
   MyForm(Owner) {
      super(Owner);

      var scroll = new RScrolledWindow(this);
      scroll.VScrollPolicy = scroll.HScrollPolicy =
          POLICY_AUTOMATIC;
      scroll.Show();

      var treeview = new RTreeView(scroll);
      treeview.Model = MODEL_LISTVIEW;
      treeview.AddColumn("Icon", IMAGE_COLUMN);
      treeview.AddColumn("Progress", PERCENT_COLUMN);
```

```
        treeview.AddColumn("Caption", NORMAL_COLUMN);
        treeview.AddColumn("Description", MARKUP_COLUMN);
        treeview.AddColumn("Radio", RADIO_COLUMN);
        treeview.AddColumn("Check", CHECK_COLUMN);
        // handle row activated event
        treeview.OnRowActivated = this.RowActivated;
        treeview.Show();

        var image1 = new RImage(null);
        // load resource - small size (1)
        image1.LoadResource("gtk-about", 1);

        var image2 = new RImage(null);
        image2.LoadResource("gtk-print", 1);

        for (var i=0;i<10;i++) {
            if (i%2)
                treeview.AddItem([image1, i/9*100, "Item ${i+1}",
                    "Some <b>kind</b> of description", true, false]);
            else
                treeview.AddItem([image2, i/9*100, "Item ${i+1}",
                    "Some <i><span color='red'>other</span> kind</i>
                    of description", false, true]);
        }
        // select the third element
        treeview.RowIndex=2;
    }

    // in Sender we have treeview
    RowActivated(Sender, EventData) {
        // get the selected item
        var item=Sender.Items[EventData];
        if (item) {
            item[3] = "<b>Clicked</b>";

            // inver the check buttons
            item[4] = !item[4];
            item[5] = !item[5];

            if (item[1] < 100) {
                item[1]++;
                // update the item on the client
                Sender.UpdateItem(EventData, item);
            } else
                Sender.DeleteItem(EventData);
```

```
       }
    }
}

class Main {
    function Main() {
           try {
               var Application=new CApplication(new MyForm(null));
               Application.Init();
               Application.Run();
               Application.Done();
           } catch (var Exception) {
               echo Exception;
           }
    }
}
```

Figure 8.11 shows the output.



Icon	Progress	Caption	Description	Radio	Check
🖶	0 %	Item 1	Some *other kind* of description	○	☑
☆	11 %	Item 2	Some **kind** of description	⊙	□
🖶	23 %	Item 3	**Clicked**	⊙	□
☆	33 %	Item 4	Some **kind** of description	⊙	□
🖶	44 %	Item 5	Some *other kind* of description	○	☑
☆	55 %	Item 6	Some **kind** of description	⊙	□
🖶	66 %	Item 7	Some *other kind* of description	○	☑
☆	77 %	Item 8	Some **kind** of description	⊙	□
🖶	88 %	Item 9	Some *other kind* of description	○	☑
☆	100 %	Item 10	Some **kind** of description	⊙	□

Figure 8.11:
listviews.con output

Note that on mobile platforms, the actual number of columns may be limited to one image, one progress bar, two text columns, and have different layout setup than desktop versions.

RIconView fires the *OnItemActivated* event, if handled, when user selects an item. The EventData string contains the selected item index in order to avoid additional network traffic by reading *RowIndex* property (which holds the same value). *RIconView* has the *Path* string property that has the same value with *RowIndex*(number). I suggest you use *RowIndex* for consistency (RTreeView, RIconView, RComboBox and REditComboBox implement *RowIndex*).

For *RIconView* you must set the *ImageColumn* property(column index) and the *TextColumn* or *MarkupColumn*. Note that you cannot add MARKUP_COLUMN flag or EDITABLE_COLUMN to an RIconView.

The previous list view example can be rewritten to use the *RIconView*.

iconviews.con

```
include Application.con
include RForm.con
include RIconView.con
include RScrolledWindow.con
include RImage.con

class MyForm extends RForm {
    MyForm(Owner) {
        super(Owner);

        var scroll = new RScrolledWindow(this);
        scroll.VScrollPolicy = scroll.HScrollPolicy =
            POLICY_AUTOMATIC;
        scroll.Show();

        var iconview = new RIconView(scroll);
        iconview.AddColumn("Icon", IMAGE_COLUMN);
        iconview.AddColumn("Description");
        iconview.ImageColumn = 0;
        iconview.MarkupColumn = 1;
        iconview.OnItemActivated = this.ItemActivated;
        iconview.Show();

        var image1 = new RImage(null);
        image1.LoadResource("gtk-about", 1);

        var image2 = new RImage(null);
        image2.LoadResource("gtk-print", 1);
        for (var i=0;i<10;i++) {
            if (i%2)
                iconview.AddItem([image1, "Item <b>${i+1}</b>"]);
            else
                iconview.AddItem([image2, "Item <b>${i+1}</b>"]);
        }
    }

    ItemActivated(Sender, EventData) {
        // Sender is the icon view
        var item=Sender.Items[EventData];
        if (item) {
            item[1]="<b>Clicked!</b>";
            Sender.UpdateItem(EventData, item);
        }
    }
}

class Main {
    function Main() {
            try {
                var Application=new CApplication(new MyForm(null));
                Application.Init();
                Application.Run();
                Application.Done();
            } catch (var Exception) {
                echo Exception;
            }
    }
}
```

Figure 8.12 shows the output.

Combo boxes use a similar programming models as list views and icon views, the only difference is that they display only the selected item. Also, item columns are nameless and are added with *AddColumns(column_number, column_type = NORMAL_COLUMN)*.

REditComboBox inherits RComboBox and has most of the REdit properties. The link between REdit and RComboBox is made via the *REditComboBox.TextColumn* property. Note that that the text column is better to be added with the HIDDEN_COLUMN flag, because it will be

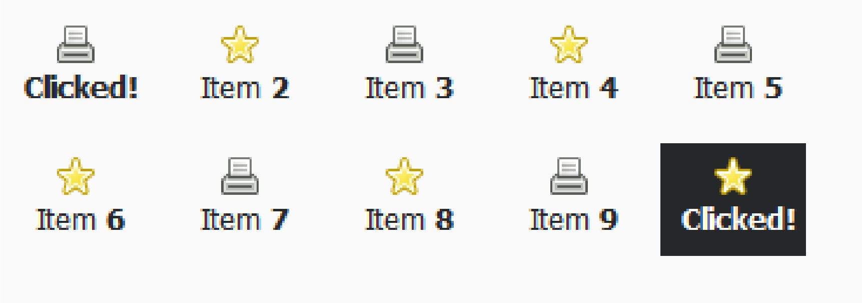


Figure 8.12:
iconviews.con output

shown twice, once as combo column and twice as text column.

comboboxes.con

```
include Application.con
include RForm.con
include RComboBox.con
include REditComboBox.con
include RImage.con
include RVBox.con

class MyForm extends RForm {
    MyForm(Owner) {
        super(Owner);

        var box = new RVBox(this);
        box.Show();

        var combobox = new RComboBox(box);
        // add one image column
        combobox.AddColumns(1, IMAGE_COLUMN);
        combobox.AddColumns(1, PERCENT_COLUMN);
        // default for AddColumn and AddColumns is TEXT_COLUMN
        combobox.AddColumns(1, MARKUP_COLUMN);
        combobox.OnChanged = this.ComboChanged;
        combobox.Show();

        var image1 = new RImage(null);
        // load resource - small size (1)
        image1.LoadResource("gtk-about", 1);
        var image2 = new RImage(null);
        image2.LoadResource("gtk-print", 1);

        var image3 = new RImage(null);
        image3.LoadResource("gtk-zoom-in", 1);

        combobox.AddItem([image1, 20, "Some <b>kind</b> of
            description"]);
        combobox.AddItem([image2, 85, "Some <i><span
            color='red'>other</span> kind</i> of description"]);
        combobox.AddItem([image3, 100, "Some <b>other</b> <i><span
            color='red'>other</span> kind</i> of description"]);
        combobox.RowIndex=0;

        var comboeditbox = new REditComboBox(box);
        comboeditbox.AddColumns(1, IMAGE_COLUMN);
        comboeditbox.AddColumns(1, HIDDEN_COLUMN);
        comboeditbox.TextColumn = 1;
        comboeditbox.AddItem([image1, "Some option"]);
        comboeditbox.AddItem([image2, "Another option"]);
        comboeditbox.AddItem([image3, "Last option"]);
        comboeditbox.RowIndex=0;
        // activate suggestions while user types
        comboeditbox.SuggestModel(comboeditbox, 1);
        comboeditbox.Text = "You can edit here";
        comboeditbox.Show();
    }

    ComboChanged(Sender, EventData) {
        var item=Sender.Items[EventData];
        if (item)
            CApplication.Message(item[2]);
    }
}

class Main {
    function Main() {
            try {
                var Application=new CApplication(new MyForm(null));
                Application.Init();
                Application.Run();
                Application.Done();
            } catch (var Exception) {
                echo Exception;
            }
    }
}
```

Figure 8.13 shows the output.

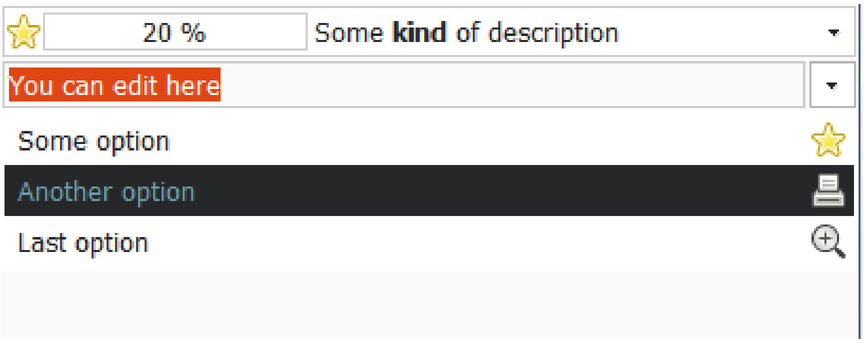


Figure 8.13:
comboboxes.con output

RTreeView(MODEL_LISTIVIEW), RIconView, RComboBox and REditComboBox have a linear structure. However, RTreeViews(MODEL_TREEVIEW) cand handle complex tree structures. *RTreeView* can also allow edit operations and some attributes for columns, rows or individual cells. The *ColumnIndex* property set/gets the index of the selected column. You could add properties to a specific cell by using *RTreeView.Columns[column_index]AddProperties(string properties)*. Available properties are:

Property name	**Notes**
foreground, cell-foreground	color as a string, eg: “#FF0000” or “red”
background, cell-background	color as a string, eg: “#FF0000” or “red”
font	font name as a string
family	font family name
language	
markup	true or false
editable	true or false
visible	true or false
strikethrough	true or false
ellipsize	real number
alignment	real number
style	
rise	
scale	real number

treviews.con

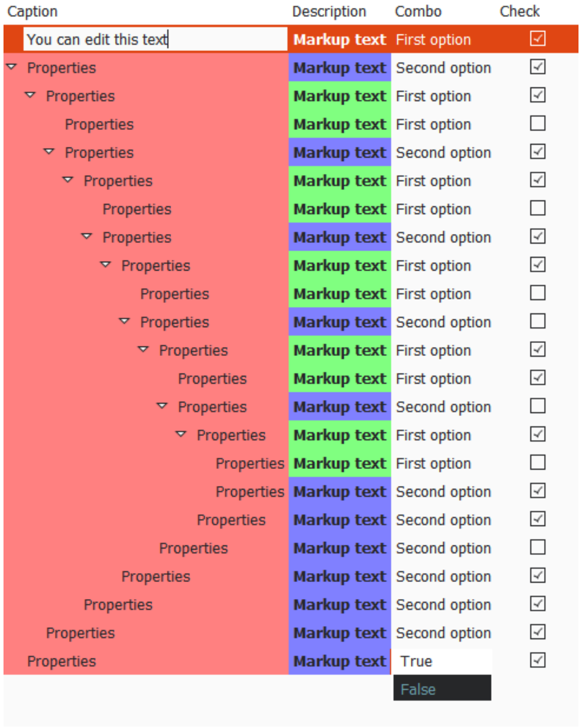
```
include Application.con
include RForm.con
include RComboBox.con
include RTreeView.con
include RScrolledWindow.con
include RImage.con

class MyForm extends RForm {
   protected var EditingCell;

   MyForm(Owner) {
      super(Owner);

      var scroll = new RScrolledWindow(this);
      scroll.VScrollPolicy = scroll.HScrollPolicy =
         POLICY_AUTOMATIC;
      scroll.Show();

      var treeview = new RTreeView(scroll);
      treeview.Model = MODEL_TREEVIEW;
      treeview.AddColumn("Caption", NORMAL_COLUMN |
         EDITABLE_COLUMN);
      // allow resize for first column
      treeview.Columns[0].Resizable=true;
      // make column light red
```

```
treeview.Columns[0].BackColor=0xFF8080;
treeview.AddColumn("Description", MARKUP_COLUMN);
treeview.Columns[1].AddAttribute("background");
treeview.AddColumn("Combo", COMBO_COLUMN | EDITABLE_COLUMN);
treeview.AddColumn("Check", CHECK_COLUMN | EDITABLE_COLUMN);

treeview.OnStartEditing = this.StartEdit;
treeview.OnEndEditing = this.EndEdit;
treeview.Show();

// create a combo model for the treeview combo column
var combobox = new RComboBox(null);
combobox.AddColumns(1);
// when we have just one column, we can send the data
// without putting it in an array
combobox.AddItem("First option");
combobox.AddItem("Second option");
combobox.AddItem("Third option");

var combobox2 = new RComboBox(null);
combobox2.AddColumns(1);
combobox2.AddItem("True");
combobox2.AddItem("False");

// Note the color attribute, "#80FF80"
// corresponding to the place in which
// the attrebute was added
treeview.AddItem(["Properties", "<b>Markup text</b>",
    "#80FF80", "First option", combobox, true]);
treeview.AddItem(["Properties", "<b>Markup text</b>",
    "#8080FF", "Second option", combobox, true]);
treeview.AddItem(["Properties", "<b>Markup text</b>",
    "#8080FF", "True", combobox2, true]);

var path=[1];
for (var i=0;i<10;i++) {
    treeview.AddChild(path, ["Properties", "<b>Markup
        text</b>", "#80FF80", "First option", combobox,
        true]);
    treeview.AddChild(path, ["Properties", "<b>Markup
        text</b>", "#8080FF", "Second option", combobox,
        true]);
    // alternatively add 1 or 0 to the path array
    path += i % 2;
}
```

```
        // expand all paths
        // you could also use Expand(path)
        treeview.ExpandAll();
    }

    // EventData contains path/column
    StartEdit(Sender, EventData) {
        EditingCell = StrSplit(EventData, "/");
    }

    EndEdit(Sender, EventData) {
        if (!EditingCell)
            return;

        var path = EditingCell[0];
        // EditingCell[1] is a string ... we need a number
        var column = value EditingCell[1];

        var item=Sender.Items[path];
        if (item) {
            // make the change in the data model
            // else it will be visible just for the user
            item[column] = EventData;
        }
    }
}

class Main {
    function Main() {
            try {
                var Application=new CApplication(new MyForm(null));
                Application.Init();
                Application.Run();
                Application.Done();
            } catch (var Exception) {
                echo Exception;
            }
    }
}
```

Figure 8.14 shows the output.

Note that when *OnStartEditing* and *OnEndEditing* are mapped, you will be notified when the user changes data in the tree view. If you don't



Caption	Description	Combo	Check
You can edit this text	**Markup text**	First option	☑
▽ Properties	**Markup text**	Second option	☑
▽ Properties	**Markup text**	First option	☑
Properties	**Markup text**	First option	☐
▽ Properties	**Markup text**	Second option	☑
▽ Properties	**Markup text**	First option	☑
Properties	**Markup text**	First option	☐
▽ Properties	**Markup text**	Second option	☑
▽ Properties	**Markup text**	First option	☑
Properties	**Markup text**	First option	☐
▽ Properties	**Markup text**	Second option	☐
▽ Properties	**Markup text**	First option	☑
Properties	**Markup text**	First option	☑
▽ Properties	**Markup text**	Second option	☐
▽ Properties	**Markup text**	First option	☑
Properties	**Markup text**	First option	☐
Properties	**Markup text**	Second option	☑
Properties	**Markup text**	Second option	☑
Properties	**Markup text**	Second option	☐
Properties	**Markup text**	Second option	☑
Properties	**Markup text**	Second option	☑
Properties	**Markup text**	Second option	☑
Properties	**Markup text**	True False	☑

Figure 8.14:
treeviews.con output

handle these events and set the EDITABLE_COLUMN flag, the changed made will not propagate to the tree view items. Every *OnStartEditing* will always be called, but *OnEndEditing* won't be fired if the user cancels the edit process. In this case *OnCancelEditing* will be called. Also, when using AddAttribute, a virtual column is created in which you must put the attribute value on the corresponding position.

OnStartEditing receives a string in the form path/column, where column is the actual item column edited(not the tree view column).
OnCancelEditing gives no information in EventData, and *OnEndEditing* has the new cell value as EventData.

8.9 Web views

The RWebView object manager a web browser view surface. On most platforms it uses the WebKit library and it has similar behavior on most platforms(mobiles and desktops). You can load URL's or HTML documents as string, you may cache images into the browser cache and you can hijack link clicks from the user. It is possible to use html/css interfaces to your application if you're familiar with these technologies.

Concept Application Server comes with a full browser examples, called WKB (found in Samples/WKB/WebKitTest.con).

You can load an URI by setting the *URI* property(string), or you can load a by setting the *Content* property(string). When the *OnNavigationRequested* event is mapped, you must handle explicitly the navigation requests. EventData will contain the request URI. You can use the Cache(string element_name, string element_content) to cache images on the client.

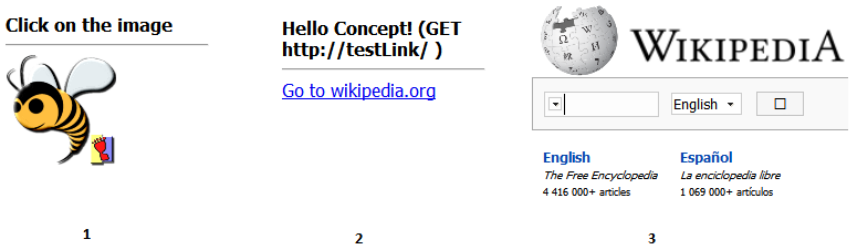
webviews.con

```
include Application.con
include RForm.con
include RWebView.con
include RImage.con

class MyForm extends RForm {
    MyForm(Owner) {
        super(Owner);

        var webview = new RWebView(this);
        // add image to client cache
        webview.Cache("concept.png",
            ReadFile("conceptclienticon.png"));
        webview.Content =
            "<strong>Click on the image</strong><hr/>"+
            "<a href='http://testLink/'>"+
            "<img border='0' src='concept.png' width='100px'
                height='100px' />"+
            "</a>";
        // to handle our "special" link
        webview.OnRequest2 = this.URLRequest;
        webview.Show();
    }

    URLRequest(Sender, EventData) {
        //EventData is "GET http://testLink/\r\n"
        var link=trim(StrSplit(EventData, " ")[1]);

        if (link=="http://testLink/")
            Sender.Content = "<strong>Hello Concept!
                (${xmlURIUnescapeString(EventData)})</strong><hr/><a
                href='http://wikipedia.org'>Go to wikipedia.org</a>";
        else
        if (!Sender.URI) {
            // open the URI
            Sender.URI=link;
        }
    }

}

class Main {
    function Main() {
            try {
                var Application=new CApplication(new MyForm(null));
                Application.Init();
                Application.Run();
                Application.Done();
            } catch (var Exception) {
                echo Exception;
            }
    }
}
```

Figure 8.15 shows the output.

8.10 Clipboard

The clipboard is a software facility used for short-term data storage and/or data transfer between documents or applications, via copy and paste operations. It is most commonly a part of a GUI environment and is usually implemented as an anonymous, temporary data buffer that can be accessed from most or all programs within the environment via defined



Click on the image

Hello Concept! (GET http://testLink/)

Go to wikipedia.org

Figure 8.15:
webviews.con output

programming interfaces[1].

The Concept UI Framework provides the *RClipboard* class, defined in RCliboard.con, for accessing the clipboard of the remote client. Note that the clipboard access is supported on all desktop clients, but may not be supported on specific mobile clients.

The RClipboard class has only static members (no need to instantiate a clipboard object). The members are:

Copy (data)
Copies data to the clipboard. The data may be a string, a number or an RImage object.

Clear ()
Clears the clipboard content buffer.

string GetImageBuffer (format="png")
Returns the image buffer in the clipboard, in the given *format*. If no image is stored on the clipboard, it will return an empty string.

string GetText ()
Returns the text buffer in the clipboard. If no text is stored on the clipboard, it will return an empty string.

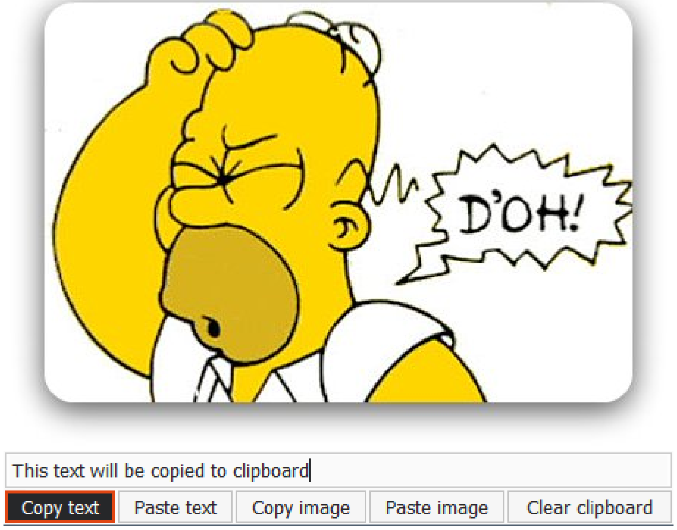
The following example illustrates all of the *RCliboard*'s functions, in a minimal example.

[1] http://en.wikipedia.org/wiki/Clipboard_(computing) on January 28, 2014

ClipboardExmaple.con

```
include Application.con
include RImage.con
include RVBox.con
include RHBox.con
include RButton.con
include REdit.con
include RClipboard.con

class MyForm extends RForm {
    var image;
    var edit;

    CopyTextClicked(Sender, EventData) {
        RClipboard::Copy(edit.Text);
    }

    PasteTextClicked(Sender, EventData) {
        edit.Text=RClipboard::GetText();
    }

    CopyImageClicked(Sender, EventData) {
        RClipboard::Copy(image);
    }

    PasteImageClicked(Sender, EventData) {
        var data=RClipboard::GetImageBuffer();
        if (data) {
            WriteFile(data,"cliptest.png");
            image.Filename="cliptest.png";
        }
    }

    ClearClipboardClicked(Sender, EventData) {
        RClipboard::Clear();
    }

    MyForm(owner) {
        super(owner);
        var vbox = new RVBox(this);
        vbox.Show();
        image = new RImage(vbox);
        image.Show();
        edit = new REdit(vbox);
        edit.Show();

        var hbox = new RHBox(vbox);
        hbox.Show();

        var copyTextButton = new RButton(hbox);
        copyTextButton.Caption = "Copy text";
        copyTextButton.OnClicked = this.CopyTextClicked;
        copyTextButton.Show();

        var pasteTextButton = new RButton(hbox);
        pasteTextButton.Caption = "Paste text";
        pasteTextButton.OnClicked = this.PasteTextClicked;
        pasteTextButton.Show();

        var copyImageButton = new RButton(hbox);
        copyImageButton.Caption = "Copy image";
        copyImageButton.OnClicked = this.CopyImageClicked;
        copyImageButton.Show();

        var pasteImageButton = new RButton(hbox);
        pasteImageButton.Caption = "Paste image";
        pasteImageButton.OnClicked = this.PasteImageClicked;
        pasteImageButton.Show();

        var clearClipboardButton = new RButton(hbox);
        clearClipboardButton.Caption = "Clear clipboard";
        clearClipboardButton.OnClicked = this.ClearClipboardClicked;
        clearClipboardButton.Show();
    }
}


class Main {
    Main() {
        try {
            var Application=new CApplication(new MyForm(NULL));
            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            //echo Exception.Information;
        }
    }
}
```

The output should look like figure 8.16, depending on the used image.



This text will be copied to clipboard

Figure 8.16: ClipboardExmaple.con output

8.11 Language and localization

The Concept Client itself is language-independent. When using stock buttons, the button layout and captions will be available to the user according to the regional settings. However, you may need to support multiple languages at your application level. For this, Concept Framework provides a very simple and efficient class, called *Lang* that enables you to write language-independent applications. Lang is a persistent key-value system, that usually stores it's elements in an XML file called "string.table.xml", located in the same directory with your application main source.

Let's look to a sample XML file first:

```
<?xml version="1.0"?>
<StringTable>
```

```
  <default langid="de"/>
  <string msgid="Name">
    <lang id="en">Name</lang>
    <lang id="de">Name</lang>
  </string>
  <string msgid="Surname">
    <lang id="en">Surname</lang>
    <lang id="de">Nachname</lang>
  </string>
  <string msgid="Invalid data was entered">
    <lang id="en">Invalid data was entered</lang>
    <lang id="de">
     Ungltige  Daten eingegeben</lang>
  </string></StringTable>
```

For each string message in your application, you must have at least one <lang> key.

Lang implements the << operator, that processes the given text according to the selected language.

Let’s consider a code block using the above example:

```
MyForm(Owner) {
    super(Owner);

    // this will automatically load the
    // string.table.xml file
    var S = new Lang();
    // we can set S.Language = "de";
    // or we can set in the XML file
    // <default langid="de"/>
    var labelName = new RLabel(this);
    // we just add S << before the actual caption
    labelName.Caption = S << "Invalid data was entered";
    labelName.Show();
}
```

When *S.Language* or *default langid* is set to “de”, on the user’s screen will display “Ungltige Daten eingegeben” instead of “Invalid data was entered”. If an undefined language or key were used with S <<, the actual key will be used. For example Lang ¡¡ “This is not defined” will return“This is not defined”. You can then use the GeoIP library to find out from where did a user connect to your application and set the language identifier accordingly.

8.12 Latency compensation

Latency, or lag, is a delay between an user action and the server response, usually caused by network problems like congestions or retransmissions.

Starting with Concept 4.0 a latency compensation protocol was added. This is based on compiling Concept function to a simplified 3-register code, that could be easily executed both by the client, simultaneously with the server. This code, is interpreted by native clients and converted to JavaScript by the JS client.

The protocol is activated per-object. For example, for compensating a button click, the **Compensated** property must be set to true.

Note that the Compensated property must be set BEFORE mapping any event for the given button.

The client may choose to stop the client-side execution if it misses some data, or if detects that an instruction is conditioned by the server.

A simple latency compensation example:

```
include Application.con
include RVBox.con
include RButton.con
include RTextView.con
include RLabel.con

import standard.lib.thread
import standard.C.math

class MyForm extends RForm {
    var edit;
    var button;

    // not this function
    OnClick(Sender, EventData) {
        var v = value edit.Text;
        button.Caption = "" + v * v;
    }

    MyForm(owner) {
        super(owner);
        var vbox = new RVBox(this);
        vbox.Show();

        edit = new RTextView(vbox);
        edit.Tooltip = "Enter a value";
        edit.Show();

        button = new RButton(vbox);
        // enabled compensation for button
        button.Compensable = true;
        button.MinHeight = 50;
        button.Caption = "Press to compute square";
        // note that Compensable was set BEFORE
        // setting the event handler
        button.OnClicked = this.OnClick;
        button.Show();
    }
}

class Main {
    Main() {
        try {
            var Application=new CApplication(new MyForm(NULL));

            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            echo Exception;
        }
    }
}
```

Now, the application simply does what it has to do. It computes the square of the given value. When using latency compensation, the value is computed immediately by the client, and by the server. In case the values would be different, the server value has priority.

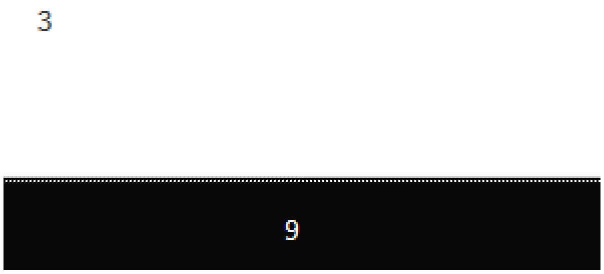


Figure 8.17:
Compensation - immediately after button click

For better understanding, let's insert a 3 seconds delay. The **IsClient()** function is the only function with different behavior on server and client. On client it returns true, on server it returns false.

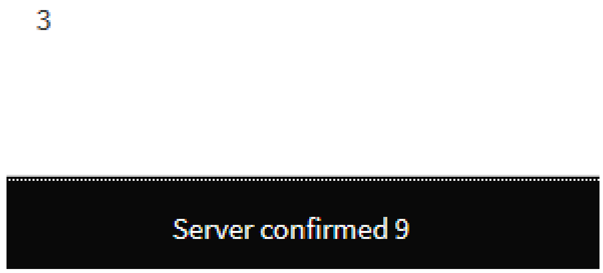
The new OnClick function will look like this:

```
OnClick(Sender, EventData) {
    var v = value edit.Text;
    if (IsClient()) {
        button.Caption = "" + v * v;
    } else {
        Sleep(3000);
        button.Caption = "Server confirmed " + v * v;
    }
}
```

The output is shown if figure 8.17, respectively 8.18.

If some function data needs to be hidden, the **protect()** function may be used. Considering the previous example, the *OnClick* function may be rewritten as:

```
OnClick(Sender, EventData) {
    var v = value edit.Text;
    button.Caption = "Server confirmed " + v * v;
    protect();
    // this code is never visible for the client
    var password = "I am to lazy to write a better example";
    // do something with password.
}
```



3

Server confirmed 9

Figure 8.18:
Compensation - 3 seconds later

```
}
```

If the **protect()** function wasn't called, the password would be visible to the client.

*Note that when using **Compensable** is set to true, the used event handler function will be compiled and sent to the client, together with all the functions used. For example, if OnClick would call a function called foo, both functions will be sent to the client. Also, if foo would call other functions, those will be also compiled and sent.*

Keep in mind that the client may choose to stop the executing at any point. For example, at the time writing the book, static functions are not supported in client.

This engine is supported on all Concept 4.0 clients, Concept Client 3.0, and Concept Client Mobile native clients (iOS 7 or greater and Android 4).

8.13 Asynchronous UI processing

Asynchronous UI processing will give the user a better feel of the application. It will seem to respond faster and allows the user to abort an operation. Concept 4.0 provides a number of mechanisms like green threads and polling to avoid blocking calls and native threads.

Every UI application has a main loop, that dispatches the received messages. Most of the time, this loop is idle. When is idle, it may call a number of functions called *Loopers*.

A looper is a delegate automatically called by the main loop. It runs in the same thread, so it should exit as fast as possible.

number RegisterLooper (delegate)
Register a delegate to run in the main loop. Returns a delegate id, to be used with UnregisterLooper.

UnregisterLooper (number id)
Removes a delegate from the main loop.

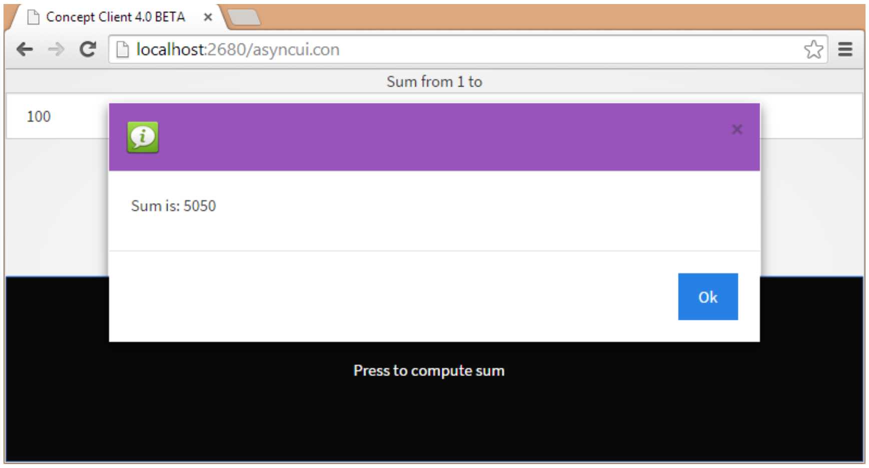
A delegate may be remove by calling *UnregisterLooper()* or by simply returning **true** when is called.

It is important to avoid making blocking calls in a looper. A blocking call will stall the main loop, freezing the application.

Note that loopers run with a low priority (about 200 calls/second). You should avoid having a large number of loopers, because it may create delays in UI. A reasonable number is 10 with a recommended maximum to 50. There are no technical limitation for loopers, but after about 100 it will cause delay in application responsiveness.

asyncui.con

```
include Application.con
include RVBox.con
include RButton.con
include REdit.con
include RLabel.con

import standard.lib.thread
import standard.C.math

class SumLooper {
   var to;
   var index = 1;
   var result = 0;

   SumLooper(to) {
        this.to = to;
        // register looper
        RegisterLooper(this.Loop);
    }

    Loop() {
        // check if finished
        if (index > to) {
            CApplication::Message("Sum is: ${this.result}");
            return true;
        }

        result += index++;
    }
}

class MyForm extends RForm {
    var vbox;
    var edit;
    var button;

    OnClick(Sender, EventData) {
        new SumLooper(value edit.Text);
    }

    MyForm(owner) {
        super(owner);
        vbox = new RVBox(this);
        vbox.Show();
        var label = new RLabel(vbox);
        label.Caption = "Sum from 1 to";
        label.Show();

        edit = new REdit(vbox);
        edit.Text = "100";
        edit.Tooltip = "Enter a value";
        edit.Show();

        button = new RButton(vbox);
        button.Compensable = true;
        button.Caption = "Press to compute sum";
        button.OnClicked = this.OnClick;
        button.Show();
        // You could simply do n*(n+1)/2
    }
```



Sum is: 5050

Figure 8.19:
Asynchronous UI

```
}

class Main {
    Main() {
        try {
            var Application=new CApplication(new MyForm(NULL));

            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            echo Exception;
        }
    }
}
```

The output is show in 8.19.

Chapter 9

Basic I/O

9.1 XML serialization

Concept Frameworks provides the Serializable base class. If a class inherits Serializable, it will contain two additional methods, called *Serialize* and *UnSerialize*.

Alternatively, a Concept class object can be serialized to a simple XML version, when you don't need to unserialize it, via the ToXML(class_object, short_version=false, encoding="UTF-8") static function.

Note that when you serialize an object, you serialize all its members(regardless of access). Serialization is recursive but has a cyclic reference detector, in order to avoid excessive data in the XML. If a member contains references to other class objects, those will be serialized, even if they don't extend *Serializable*.

The Serialize prototype: *Serializable.Serialize(filename="-", encoding="UTF-8")*

If filename is "-" the XML data will be written to standard output (on screen). Else, if a filename is given, it will be put to that file. If filename is empty, a string containing the XML data will be returned.

```
include Serializable.con
```

```
class Student {
    var Name="";
    var Course="";
    var[] Friends;
}

class StudentList extends Serializable {
    var[] Students;
}

class Main {
    Main() {
        var list = new StudentList();

        var student1 = new Student();
        student1.Name = "Eduard";
        student1.Course = "Concept Programming";
        list.Students += student1;

        var student2 = new Student();
        student2.Name = "Maria";
        student2.Course = "Concept Programming";
        list.Students += student2;

        var student3 = new Student();
        student3.Name = "John";
        student3.Course = "Concept Programming";
        list.Students += student3;

        student1.Friends += student2;
        student1.Friends += student3;

        list.Serialize("data.xml");
        // alternatively
        // var buffer = list.Serialize("");
    }
}
```

The resulting file:

```
<?xml version="1.0" encoding="UTF-8"?>
<object cycid="1" class="StudentList">
  <member name="Students" type="array">
    <array cycid="2">
```

```
   <element index="0" type="object">
    <object cycid="3" class="Student">
     <member name="Name" type="string">Eduard</member>
     <member name="Course" type="string">Concept
       Programming</member>
     <member name="Friends" type="array">
      <array cycid="4">
       <element index="0" type="object">
        <object cycid="5" class="Student">
         <member name="Name" type="string">Maria</member>
         <member name="Course" type="string">Concept
           Programming</member>
         <member name="Friends" type="array">
          <array cycid="6"/>
         </member>
        </object>
       </element>
       <element index="1" type="object">
        <object cycid="7" class="Student">
         <member name="Name" type="string">John</member>
         <member name="Course" type="string">Concept
           Programming</member>
         <member name="Friends" type="array">
          <array cycid="8"/>
         </member>
        </object>
       </element>
      </array>
     </member>
    </object>
   </element>
   <element index="1" type="object">
    <cyclic_reference refID="5"/>
   </element>
   <element index="2" type="object">
    <cyclic_reference refID="7"/>
   </element>
  </array>
 </member>
</object>
```

Note that every object is defined just once in the XML files. For references of the same object, you have the *cyclic_reference* tag referencing an already serialized object. This is a convenient way of storing data for your applications, because is extremely easy to restore the data from the XML file, using the *Serialize.UnSerialize(file_or_buffer, is_buffer=false, var out_err=null)* function.

Note that your application must have a definition of the actual object classes you unserialize.

```
include Serializable.con

// we must define all the serialized classes
class Student {
    var Name="";
    var Course="";
    var[] Friends;
}

class StudentList extends Serializable {
    var[] Students;
}

class Main {
    Main() {
        var list = Serializable.UnSerialize("data.xml");
        if (list) {
            var len = length list.Students;
            for (var i = 0; i < len; i++) {
                var student = list.Students[i];
                echo "Student ${student.Name} (Course:
                    ${student.Course}) was loaded and it has ${length
                    student.Friends} friends\n";
            }
        }
    }
}
```

Will output:

```
Student Eduard (Course: Concept Programming) was loaded and it has
    2 friends
Student Maria (Course: Concept Programming) was loaded and it has 0
    friends
Student John (Course: Concept Programming) was loaded and it has 0
    friends
```

Alternatively, you can use the *ToXML* static function that serializes a class object(one way), but doesn't store meta data for restoring the data structure. Note that is not mandatory for a class to extend *Serializable* for one to be able to use *ToXML*.

For the first example, if we replace:

```
        list.Serialize("data.xml");
```

with

```
        echo ToXML(list);
```

Will output:

```
<?xml version="1.0" encoding="UTF-8"?>
<StudentList>
    <Students type="array">
        <element index="0" type="Student">
            <Name type="string">Eduard</Name>
            <Course type="string">Concept Programming</Course>
            <Friends type="array">
                <element index="0" type="Student">
                    <Name type="string">Maria</Name>
                    <Course type="string">Concept Programming</Course>
                    <Friends type="array"/>
                </element>
                <element index="1" type="Student">
                    <Name type="string">John</Name>
                    <Course type="string">Concept Programming</Course>
                    <Friends type="array"/>
                </element>
            </Friends>
        </element>
        <element index="1" type="Student">
            <Name type="string">Maria</Name>
            <Course type="string">Concept Programming</Course>
            <Friends type="array"/>
        </element>
        <element index="2" type="Student">
```

```
            <Name type="string">John</Name>
            <Course type="string">Concept Programming</Course>
            <Friends type="array"/>
        </element>
    </Students>
</StudentList>
```

This is useful when exporting data to be used with other systems. By using *ToXML* function, you avoid managing complex XML structures.

This can be simplified further more, by setting the *short_version* parameter to true. This will skip all the meta data associated with the data fields.

```
        echo ToXML(list, true);
```

Will output:

```
<?xml version="1.0" encoding="UTF-8"?>
<StudentList>
  <Students>
    <Student>
      <Name>Eduard</Name>
      <Course>Concept Programming</Course>
      <Friends>
        <Student>
          <Name>Maria</Name>
          <Course>Concept Programming</Course>
          <Friends/>
        </Student>
        <Student>
          <Name>John</Name>
          <Course>Concept Programming</Course>
          <Friends/>
        </Student>
      </Friends>
    </Student>
    <Student>
      <Name>Maria</Name>
      <Course>Concept Programming</Course>
      <Friends/>
    </Student>
    <Student>
      <Name>John</Name>
      <Course>Concept Programming</Course>
      <Friends/>
    </Student>
  </Students>
</StudentList>
```

The *standard.lang.serialization* static import libraries also adds some convenient static function for accessing data members by their name.

```
boolean HasMember(object obj, string member_name)
boolean GetMember(object, obj, string member_name, var value[,
    as_delegate=false])
boolean SetMember(object obj, string member_name, new_value)
object CreateObject(string class_name)
```

The *HasMember* function returns true if obj has a member called *member_name*.

GetMember sets the *value* parameter to the value of the member called *member_name*, and returns true if succeeded. If the *as_delegate* parameter is set to true, and textitmember_name will be a function member name, this function will set value to a delegate referencing the function. If is false, the function will be evaluated (assuming that it takes no parameters).

The *SetMember* function will try to set the member called *member_name* to the given *new_value*. If succeeded, it will return true.

The *CreateObject* will try to instantiate a class called *class_name* and return an object of that class (assuming that no constructor is defined or if defined, it takes no parameters). If the class is not defined, it will return *null*.

Another useful function is *ToArray*:

```
array ToArray(object)
```

This function converts an object to a key-value array, having member names as keys and member values as values, for example:

```
import standard.lang.serialize
```

```
class Student {
    var Name = "";
    var Course = "";
    var[] Friends;
}

class Main {
    Main() {
        var student = new Student();
        student.Name = "Maria";
        student.Course = "Mathematics";
        student.Friends = ["John", "Anna"];
        echo ToArray(student);
    }
}
```

The output:

```
Array {
        [0,"Name"] => Maria
        [1,"Course"] => Mathematics
        [2,"Friends"] =>
               Array {
                       [0] => John
                       [1] => Anna
               }
}
```

9.2 JSON serialization

JSON, or JavaScript Object Notation, is an open standard format that uses human-readable text to transmit data objects consisting of attributevalue pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML. Although originally derived from the JavaScript scripting language, JSON is a language-independent data format, and code for parsing and generating JSON data is readily available in a large variety of programming languages, including Concept.

Concept module *standard.lib.json* define two static functions:

```
string JSONSerialize(object, number array_as_objects=false);
```

and

```
array JSONDeserialize(string json_data);
```

JSONSerialize returns the coresponding JSON string associated with the given object. The serialization works exactly as with XML, except that reference links is not available. Special attention should be payed to cyclic references, which can cause a stack overflow or infinite recursion. When setting the array_as_objects flag, the key-value arrays will be serialized as if they were a class object. The *object* parameter can be a class object or an array, unlike XML which is limited to objects.

Upon deserialization, the initial object cannot be restored, because JSON simply doesn't keep any meta-data. Instead, an object representation as a key-value array will be obtained. This has the advantage that the deserializer program may not have a definition for the initial object class, like in the case of XML serialization.

```
import standard.lib.json

class Student {
    var Name="";
    var Course="";
    var[] Friends;
}

class Main {
    Main() {
        var list = new [];

        var student1 = new Student();
        student1.Name = "Eduard";
        student1.Course = "Concept Programming";
        list += student1;

        var student2 = new Student();
        student2.Name = "Maria";
        student2.Course = "Concept Programming";
        list += student2;
        var student3 = new Student();
        student3.Name = "John";
        student3.Course = "Concept Programming";
        list += student3;

        student1.Friends += student2;
        student1.Friends += student3;

        var json = JSONSerialize(list);
        echo json;
    }
}
```

Will produce:

```
[
    {
        "Name": "Eduard",
        "Course": "Concept Programming",
        "Friends": [
                      {
                          "Name": "Maria",
                          "Course": "Concept Programming",
                          "Friends": [ ]
                      },
                      {
                          "Name": "John",
                          "Course": "Concept Programming",
                          "Friends": [ ]
                      }
                  ]
    },
    {
        "Name": "Maria",
        "Course": "Concept Programming",
        "Friends": [ ]
    },
    {
        "Name": "John",
        "Course": "Concept Programming",
        "Friends": [ ]
    }
]
```

If in the previous example we'd run:

```
echo JSONDeserialize(json);
```

Will obtain a Concept array:

```
Array {
    [0] =>
        Array {
            [0,"Name"] => Eduard
            [1,"Course"] => Concept Programming
            [2,"Friends"] =>
                Array {
                    [0] =>
                        Array {
                            [0,"Name"] => Maria
                            [1,"Course"] => Concept Programming
                            [2,"Friends"] =>
                                Array {
                                }
                        }
                    [1] =>
                        Array {
                            [0,"Name"] => John
                            [1,"Course"] => Concept Programming
                            [2,"Friends"] =>
                                Array {
                                }
                        }
                }
        }
    [1] =>
        Array {
            [0,"Name"] => Maria
            [1,"Course"] => Concept Programming
            [2,"Friends"] =>
                Array {
                }
        }
    [2] =>
        Array {
            [0,"Name"] => John
            [1,"Course"] => Concept Programming
            [2,"Friends"] =>
            Array {
            }
      }
}
```

In practice, JSON is useful when writing or interacting with http/web services. JSON is preferred in web service because of its lightweight and its focus on the data, instead the data and form, like XML.

9.3 Binarization

Similar to serialization, Concept provides "binarization" API. Binarization is the representation of an object or array to a binary buffer. This is more efficient than XML or JSON serialization, both in terms of CPU usage and memory.

Internally the data is stored using the following structures:

varsize

```
8 bit to 72 bit
if size <= 0x7D
    store size on 8 bits
else
if size <= 0xFFFF
    write 0x7E (8 bits)
    write size on 16 bits (big endian)
else
    write 0x7F (8 bits)
    write size on 64 bits (big endian)
endif
```

string

```
variablesize string_size + string_buffer
```

variable

```
8 bit type
case type:
    string (0x03)
```

```
                string val
        number (0x02)
                64bit_double val
        object (0x04)
                object
        array (0x05)
                array
        delegate (0x06)
                string class
                string member
end case
```

object

```
32 bit: object id (if object id is negative, link to abs(object id). end)
string classname
while (string classmember (size ¿ 0))
        variable
```

array

```
32 bit: object id (if object id is negative, link to abs(object id). end)
variablesize size
foreach element
        string key
        variable
endfor
```

string BinarizeObject (obj_or_array, mode = 0)

Returns a string buffer containing the binary version of the object or array. If *mode* is 0, all members will be binarized. If mode is 1, members having the default value will be omitted. For example:

```
class A {
    var some_member = 1;
    var some_other_member = 2;

    A(val) {
        some_other_member = val;
    }
}
...
    BinarizeObject(new A(3), 1);
...
```

will cause *some_member* to be skipped when binarizing, because it has the default value, and no function or constructor set it. This may be more efficient in resource usage. Any non-zero mode will skip the default members. Additionally, *mode* **2** will also skip array members, *mode* **3** will skip objects, *mode* **4** will skip both array and objects and *mode* **5** will skip UI vectors (used internally by Concept Framework). The function will return a string buffer representation of *obj_or_array*.

object UnBinarizeObject (string buffer, offset = 0, filter = [ ])
Restores the Concept objects from a buffer returned by *BinarizeObject*. If *offset* is set, the function will parse the buffer starting at the given offset. Additionally, if *filter* is set, only object members contained in the list will be un-binarized. For example (see previous for class A definition), **UnBinarizeObject***(buffer, 0, [”some_member”])* will return an A object with only some_member initialized. If succeeded, the function will return a reconstructed object obtained from the given input buffer.

```
import standard.lang.serialize

class Message {
    var m = "hello";

    Message(m) {
        this.m = m;
    }
}

class Main {
    Main() {
        var msg = new Message("Hello world!");
        var buffer = BinarizeObject(msg);
        var buffer_xml = SerializeObject(msg);
        echo "Binary version: " + length buffer + " bytes\n";
        echo "XML version: " + length buffer_xml + " bytes\n";
        var recv_msg = UnBinarizeObject(buffer);
        echo recv_msg.m;
    }
}
```

Output:

```
Binary version: 30 bytes
XML version: 116 bytes
Hello world!
```

As you can see, the binary representation of the object is almost 4 times smaller than XML.

Similar to the Serialization base class, you can also use **Binarizable** class, by including Binarizable.con.

The Binarize prototype: *Binarizable.Binarize()*

The UnBinarize prototype: *Binarizable.UnBinarize(var bufffer, number offset = 0)*

Using these two methods, the XML serialization example could be easily modified to ”binarize” instead of XML serialize:

```
include Binarizable.con
import standard.C.io

class Student {
    var Name="";
    var Course="";
    var[] Friends;
}

class StudentList extends Binarizable {
    var[] Students;
}

class Main {
    Main() {
        var list = new StudentList();

        var student1 = new Student();
        student1.Name = "Eduard";
        student1.Course = "Concept Programming";
        list.Students += student1;

        var student2 = new Student();
        student2.Name = "Maria";
        student2.Course = "Concept Programming";
        list.Students += student2;

        var student3 = new Student();
        student3.Name = "John";
        student3.Course = "Concept Programming";
        list.Students += student3;

        student1.Friends += student2;
        student1.Friends += student3;

        WriteFile(list.Binarize(), "students.bin");
    }
}
```

9.4 Compression

For even more memory efficiency in binarization, compression may be used. The import library standard.lib.serialize provides two straight forward APIs for compressing and decompressing string buffers.

string compress (string buffer, level = -1)
Returns the compressed version of the input *buffer*. *level* sets the level of compression, between 0 (no compression) and 10 (maximum compression).

string uncompress (string buffer)
Uncompresses the given buffer and returns the original buffer. If *buffer* is not valid or corrupted, an empty string will be returned.

The previous binarization example could be modified to use compression:

```
import standard.lang.serialize

class Message {
    var m = "hello";

    Message(m) {
        this.m = m;
    }
}
```

```

class Messages {
    var[] Messages;

    operator += (m) {
        Messages += m;
    }
}

class Main {
    Main() {
        var msg = new Messages();
        for (var i = 0; i < 100; i++)
            msg += new Message("Hello world!");
        var buffer = BinarizeObject(msg);
        var buffer_compress = compress(buffer);
        var buffer_xml = SerializeObject(msg);
        echo "Binary version: " + length buffer + " bytes\n";
        // note the size of the buffer
        echo "Compressed version: " + length buffer_compress + "
            bytes\n";
        echo "XML version: " + length buffer_xml + " bytes\n";
        var recv_msg = UnBinarizeObject(uncompress(buffer));
    }
}
```

Output:

```
Binary version: 3130 bytes
Compressed version: 271 bytes
XML version: 14122 bytes
```

Keep in mind that compression/decompression will have an impact on the CPU usage of your application. It is important to put it in balance: CPU vs Memory or bandwidth or storage.

9.5 Files

Files are relatively easy to manage in Concept. A total of 3 native interfaces are available.

Lazy programmer interface

this is the simplest way to access relatively small files using ReadFile(returns file content) and WriteFile(returns true if succeeded, false if failed) static functions.

```
string ReadFile(string filename);
number WriteFile(string content, string filename);
```

File class

The recommended way of dealing with large files

C-style low-level interface

C file I/O functions ported in Concept. These function include *fopen, fclose, fseek, fread, fwrite* and many more. See Concept Documentation for static library standard.C.io for the complete list of supported functions.

ReadFile simply loads a file into a string buffer, and *WriteFile* writes a string buffer into the given file. If the file exists, it will be overwritten.

The File class is a wrapper of C I/O functions that can efficiently handle files of any size. For opening a file, first you must set the *Name* property to point to the given file. Then, a call to *Open()* must be made. If the file was successfully opened, *Open* will return true.

Reading method	**Return**
Read(var buffer, max_size)	the number of bytes read, or -1 on error
GetChar()	the next character or empty string on error
GetString(separator="", max=0xFF)	the next line in file or empty string on error

Write is done via *Write(string buffer)* method, returning the bytes written or -1 on error. All these methods throws simple static exception (strings describing the error).

It is important to set the correct mode by setting *Mode* property(see table), before calling Open.

File modes as used by C fopen function:

Mode	Notes
“r”	**read:** Open file for input operations. The file must exist.
“w”	**write:** Create an empty file for output operations. If a file with the same name already exists, its contents are discarded and the file is treated as a new empty file.
“a”	**append:** Open file for output at the end of a file. Output operations always write data at the end of the file, expanding it. Repositioning operations (*Seek* or C’s *fseek*) are ignored. The file is created if it does not exist.
“r+”	**read/update:** Open a file for update (both for input and output). The file must exist.
“w+”	**write/update:** Create an empty file and open it for update (both for input and output). If a file with the same name already exists its contents are discarded and the file is treated as a new empty file.
“a+”	**append/update:** Open a file for update (both for input and output) with all output operations writing data at the end of the file. Repositioning operations (*Seek*, *fseek*) affects the next input operations, but output operations move the position back to the end of file. The file is created if it does not exist.

The *EOF* read-only property returns true when the end-of-file indicator is set.

```
include File.con

class Main {
    Main() {
        if (!WriteFile("The quick brown fox jumps over the lazy
            dog", "out.txt"))
            return -1;

        var file = new File();
        // choose file
        file.Name = "out.txt";
        try {
            if (file.Open()) {
                var res = file.Read(var buffer, 0xFF);
                if (res > 0)
                    echo buffer;
                file.Close();
            }
        } catch (var exc) {
            echo "Errror: "+exc;
        }
        // delete the created file
        _unlink("out.txt");
    }
}
```

Outputs:

```
The quick brown fox jumps over the lazy dog
```

We could replace the above functions with File.Write and ReadFile resulting in the same output(simplified version, exception handling):

```
include File.con

class Main {
    Main() {
        // set mode to write
        var file = new File("wb");
        file.Name = "out.txt";
        if (file.Open()) {
            file.Write("The quick brown fox jumps over the lazy
                dog");
            file.Close();
            // read and print the file contents
            echo ReadFile("out.txt");
            _unlink("out.txt");
        }
    }
}
```

Seek(number offset, origin=SEEK_SET) function changes the file position indicator to *offset* bytes from origin. Origin can be SEEK_SET(beginning of the file), SEEK_CUR(current position) or SEEK_END(end of file). *Tell()* function returns the actual position indicator.

9.6 Listing directories

The *DirectoryList* class provides access to directory files. The class implements two methods:

```
static array DirectoryList::ListByType(string directory, number
    type);
static array DirectoryList::ListByExt(string directory, array
    extensions, no_dir = true);
```

type must be one of the following values:

S_IFREG
regular file

S_IFBLK
block device

S_IFDIR
directory

S_IFCHR
character device

S_IFIFO
FIFO/pipe

ListByType will return an array containing all the files of the given type in the *directory*.

ListByExt will return an array containing all the files having the given extension(s). If you want to list all the files in a directory, you can call:

```
var file_list = DirectoryList.ListByExt(directory, ["*"]);
```

For example, if you want to list *txt* and *pdf* extensions, the following call may be used:

```
var file_list = DirectoryList.ListByExt(directory, ["txt", "pdf"]);
```

When the *no_dir* parameter is set to false, the function will also return the directories for satisfying the given criteria.

Both function will return the files sorted by name. Note that sorting is case-sensitive.

```
include DirectoryList.con

class Main {
   Main() {
      echo DirectoryList.ListByExt(".", ["*"], false);
   }
}
```

The above example will show the contents of the current directory.

9.7 File system

The *IO* class, defined in IO.con provides basic file system management functions. These functions are also defined in *standard.C.io* using the same prototypes as the equivalent C functions.

IO's static members:

Error ()
Returns the the last error I/O error code (equivalent to C's errno)

ChDir (string path)
Changes the current directory to path. Returns 0 on success.

Erase (string filename)
Erases *filename*. Returns 0 on success.

RmDir (string path)
Removes the *path*. Note that *path* must be empty. Returns 0 on success

MkDir (string path)
Creates a new directory described by path. Returns 0 on success.

Remove (string filename)
Deletes *filename*. Returns 0 on success.

Exists (string path)
Checks if file or directory exists. Returns true if the path exists, false otherwise.

Exec (string execpath, array parameters=null)
Executes execpath with the given parameters.

Stat (string path)
Obtains information about the named file and returns it as a key-value array, containing the following keys: *st_gid, st_atime, st_ctime, st_dev, st_ino, st_mode, st_mtime, st_nlink, st_rdev, st_size* and *st_uid*. The path argument points to a pathname naming a file. Read, write, or execute permission of the named file is not required. On error, it returns an empty array.

Besides the *IO* class, the IO.con file also defines the *Env* class, for accessing environment variables of the current process.

Env's static members:

string Get (string envname)
Searches the environment of the calling process (see the Base Definitions volume of IEEE Std 1003.1-2001, Chapter 8, Environment Variables) for the environment variable name if it exists and return its value. If the variable is not defined, it returns an empty string.

number Put (string varexpr)
Will use the *varexpr* argument to set environment variable values. The *varexpr* argument should be a string of the form "name=value". This function shall make the value of the environment variable name equal to value by altering an existing variable or creating a new one. On success, the function returns 0.

Both *Env* and *IO* functions are defined as static functions, so no need to create objects for these classes.

For example, if wanted to create a directory, a call to IO::MkDir could be made:

```
include IO.con

class Main {
    Main() {
        if (IO.MkDir("test"))
            echo "Error: ${strerror(IO::Error())}";
    }
}
```

This will create a directory called *test*, if not exists, in the current directory.

The error code returned by *IO::Error* can be converted to a string, by using the *strerror(number errorcode)* static function, defined in *standard.C.io*.

```
include IO.con

class Main {
    function Main() {
        var temp = Env.Get("TEMP");
        if (!temp)
            temp = Env.Get("TMP");
        echo temp;
    }
}
```

You could get the temporary directory location by reading the *TEMP* or *TMP* environment variables (depending on the operating system).

Note that *IO::RmDir* will return an error when trying to delete a directory. For successfully deleting a directory, you must first delete all its contents (files and subdirectories).

9.8 Configuration files

Concept Framework implements two static functions, IniGet and IniSet to provide access to custom configuration files. Both functions are implemented in *standard.C.io* import library.

```
string IniGet(string ini_filename, string section, string key,
    string default_value="");
boolean IniSet(string ini_filename, string section, string key,
    string value="");
```

IniGet returns the value associated with the given key in the given section in the file specified by *ini_filename*, or *default_value* if the key is not present in the given ini file.

IniSet sets the given key in the given section, in the given *ini_filename* to the given value. Note that all values must be strings. Returns true on success or false on error (for example if it has not writing rights for *ini_filename*).

This is a convenient way of storing database login settings or various kind of data, that needs to be viewed or edited by humans.

A configuration files must contain sections, enclosed by brakets [ ], and key-value pairs. It may contain comments, prefixed by “;”. Each comment ends at the line end.

configuration.ini

```
[SomeSection]
; you can add coments by prefixing them with ";"
SomeStringValue = "String value"
SomeNumberValue = 10

[SomeOtherSection]
OtherValue     =   "test"
```

A call to

```
IniGet("configuration.ini", "SomeSection", "SomeStringValue")
```

will return

```
String value
```

Note that you request SomeNumberValue, a string containing “10” will be returned.

9.9 Pipes

An anonymous pipe is a simplex FIFO communication channel that may be used for one-way interprocess communication (IPC). An implementation is often integrated into the operating system’s file IO subsystem. Typically a parent program opens anonymous pipes, and creates a new process that inherits the other ends of the pipes, or creates several new processes and arranges them in a pipeline.

Anonymous pipes can be created by using the *pipe(read_pipe, write_pipe)* defined in *standard.C.io* import library. On succeeded, pipe will return 0 and will set *read_pipe* and *write_pipe* to file descriptors. Then, simply use the *read*(file_descriptor, var buffer, number max_size) and *write*(file_descriptor, string buffer) static functions for reading and writing on the pipe. Both function return the number of bytes read, respectively written or -1 in case of error. Note that the returned descriptors are read-only, respectively write-only.

At the end, remember to close both file descriptors by calling close(read_pipe) and close(write_pipe) to avoid memory leaks.

PipeServer.con

```
import standard.C.io
import standard.lib.thread

define MAX_BUFFER 100

class Main {
    var write_pipe;

    Thread() {
        // create the child process and wait its finish
        exec("concept", "PipeChild.con", "$write_pipe");
    }

    Main() {
        if (!pipe(var read_pipe, write_pipe)) {
            // create the child thread
            RunThread(this.Thread);

            read(read_pipe, var buffer, MAX_BUFFER);
            echo buffer;

            close(read_pipe);
            close(write_pipe);
        } else
            echo "Error creating pipe";
    }
}
```

PipeClient.con

```
import standard.C.io
import standard.lang.cli

class Main {
    Main() {
        var arguments = CLArg();
        if (!arguments)
            echo "No arguments received";

        var write_pipe = value arguments[0];
        write(write_pipe, "Hello from the child process");
        close(write_pipe);
    }
}
```

The CLArg() function defined in standard.lang.cli import library, returns the parameters received by the program. This is useful for CLI (command-line interpreter) applications, where you want the user to send parameters to your application. The return value of the *Main* constructor will be sent back to the command line shell.

RunThread runs the given delegate in a separate thread. This will be discussed in the *Multi-threading* section of this book.

The previous example will print

```
Hello from the child process
```

Alternatively, a program output or input can be redirected to a pipe by using the *File* class discussed earlier, with the only difference that instead of calling *Open()*, *POpen()* (short for pipe open) will be used. Instead of setting the *File.Name* property to an actual file, we will set to the program needed whose output or input is to be redirected. Note that just the “rb” and “rb” modes are available. You can redirect only the input or only the output.

POpenServer.con

```
include File.con

class Main {
    Main() {
        // set mode to write
        var file = new File("rb");
        file.Name = "concept child2.con";
        if (file.POpen()) {
            // pipe opened!
            file.Read(var buffer, 100);
            echo buffer;
            file.Close();
        }
    }
}
```

POpenClient.con

```
class Main {
    Main() {
        echo "Hello world from a pipe application";
    }
}
```

Will output:

```
Hello world from a pipe application
```

When possible, this is the recommended way of creating pipes between processes, due to its simplicity and no actual threads are created or needed.

Note that both write and read operations on pipe can block. *write* may block if the buffer is full (the data was not read by the other process). *read* will block if the buffer is empty, until a write or a close operation will occur.

9.10 Remote files

Some time the files needed are not located on the Concept Application Server. The URL class provides access to remote files located, for example, on a HTTP server or FTP server.

URL has an extremely simple interface, independent of the network protocol. It has the following members:

Get (string url, post_array=null, cert_verification=true, robot_id=CURL_ROBOTID, no_post=false)
gets a remote file described by the url parameter. If using http, you may post variables via the *post_array* key-value array. For example, if you want to post via HTTP a variable named "query" you must set, post_array["query"] = "desired value"

IsSuccessfull ()
returns true if the previous Get call was successful (HTTP only)

IsRedirect ()
returns true if the previous Get returned a redirect. The location of the redirect can be obtain by reading the **Location** property

ContentType : read-only property
Returns the content type (HTTP only)

ContentLength : read-only property
Returns the content length in bytes

Location : read-only property
Returns the current location (HTTP only)

HTTPVersion : read-only property
Returns the current location (HTTP only)

Host : read-only property
Returns the host used by the current request

Response : read-only property
Returns the HTTP response code (HTTP only)

Headers : read-only property
Returns the HTTP headers (HTTP only)

Data : read-only property
Returns the file content as string buffer

URL is also useful in accessing REST API's or SOAP-based services. It can handle secured TLS or SSL connections.

URLExample.con

```
include URL.con

class Main {
    Main() {
        var url=new URL();
        // get the first web page in the world
        url.Get("http://info.cern.ch/hypertext/WWW/TheProject.html");
        if (url.IsSuccessful())
            echo url.Data;
        else
            echo "Error ${url.Response}!";
    }
}
```

The same example, can be modified, when accessing a file on any other protocol other than HTTP, for example FTP:

```
var url=new URL();
url.Get("ftp://user:password@your.ftp.org/file.txt");
if (url.Data)
    echo url.Data;
```

Or a local file:

```
var urlLocal=new URL();
urlLocal.Get("file:///home/eduard/file.txt");
if (urlLocal.Data)
    echo urlLocal.Data;
```

For low level access to the file transfer see the Concept Documentation for standard.net.curl import library.

9.11 CSV files

A comma-separated values (CSV) (also sometimes called character-separated values, because the separator character does not have to be a comma) file stores tabular data (numbers and text) in plain-text form. Plain text means that the file is a sequence of characters, with no data that has to be interpreted instead, as binary numbers. A CSV file consists of any number of records, separated by line breaks of some kind; each record consists of fields, separated by some other character or string, most commonly a literal comma or tab. Usually, all records have an identical sequence of fields. The Concept Framework provides the CSV class, defined in CSV.con for handling this type of file.

CSV class members:

Delim : string property (default is ’,’)
Sets the field delimiter

Quote : string property (default is ‘”’)
Sets the field enclosing character

Error : number property
Gets the last error code

ErrorString : number property
Gets the last error as a human readable string

array Parse (string data, number is_complete=false)
Parses a block of csv data (may be complete or not). If the block is not complete, only the values for the complete rows will be returned, the incomplete rows being returned on the next call. Returns an array containing the parsed rows as arrays.

static Do (string csv_buffer)
Parses the CSV buffer and return its content as an array of rows (arrays).

Done ()

Releases the CSV parser

Assuming we have the following CSV file, called **test.csv**:

```
Latitude,Longitude,Name
48.1,0.25,"First point"
49.2,1.1,"Second point"
47.5,0.75,"Third point"
```

We could read it as it follows:

```
include CSV.con
import standard.C.io

class Main {
    function Main() {
        echo CSV::Do(ReadFile("test.csv"));
    }
}
```

Output:

```
Array {
    [0] =>
        Array {
            [0] => Latitude
            [1] => Longitude
            [2] => Name
        }
    [1] =>
        Array {
            [0] => 48.1
            [1] => 0.25
            [2] => First point
        }
    [2] =>
        Array {
            [0] => 49.2
            [1] => 1.1
            [2] => Second point
        }
    [3] =>
```

```
        Array {
            [0] => 47.5
            [1] => 0.75
            [2] => Third point
        }
}
```

For large CSV files, it is recommended that you use the *Parse* method instead for the simple *CSV::Do* static function, for minimizing the memory usage.

```
[..]
// f is a File
var csv = new CSV();
// read 64k blocks
while (f.Read(var buffer, 0xFFFF)>0) {
    do_something_with_these_Records(csv.Parse(block));
}
csv.Close();
[..]
```

CSV is a simple an convenient way of exchanging data between applications, while maintaining a human-readable structure. Lots of applications can handle CSVs, like Microsoft Excel, OpenOffice.org Calc or LibreOffice Calc.

9.12 Archive files

The Concept Framework comes with support for zip files. All the zip static functions are defined in *standard.arch.zip* (see Concept Framework documentation). The Arc class, defined in Arc.con provides a convenient way of dealing with archived files.

The archive creation is strait-forward:

```
include Arc.con

class Main {
    Main() {
        var arc=Arc::Create("archive.zip");
        // a file named readme.txt must be created first
        Arc::Add(arc, "readme.txt");
        Arc::AddContent(arc, "dir/anotherfile.txt", "This is the
            file content");
        Arc::Close(arc);
    }
}
```

The unpacking is even simpler:

```
include Arc.con

class Main {
    Main() {
        Arc::UnZip("archive.zip");
    }
}
```

You can pack either existing files, or files specified by name and buffer content (see Arc::AddContent).

9.13 OCR

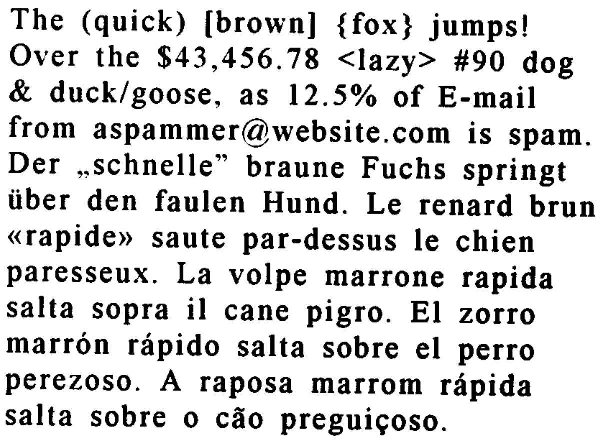
OCR(Optical Character Recognition) refers to a mechanism of identifying text in scans or images of printed text. Concept Framework offers OCR (Optical Character Recognition) support via a simple function defined in *standard.lib.ocr*. The OCR API is based on an open source library called Tesseract. The OCR function, taks two parameters: the image filename (must be an uncompressed .tif or .bmp) to extract the text from, and the output buffer. On success, the function returns 0.

The OCR function is straight forward:

ocrexample.con

```
import standard.lib.ocr

class Main {
    Main() {
        if (!OCR("eurotext.tif", var data))
```

```
           echo data;
    }
}
```



The (quick) [brown] {fox} jumps!
Over the $43,456.78 <lazy> #90 dog
& duck/goose, as 12.5% of E-mail
from aspammer@website.com is spam.
Der „schnelle" braune Fuchs springt
über den faulen Hund. Le renard brun
«rapide» saute par-dessus le chien
paresseux. La volpe marrone rapida
salta sopra il cane pigro. El zorro
marrón rápido salta sobre el perro
perezoso. A raposa marrom rápida
salta sobre o cão preguiçoso.

Figure 9.1:
eurotext.tif, used as input in ocrexample.con

The output:
The (quick) [brown] fox jumps!
Over the $43,456.78 <lazy> #90 dog
& duck/goose, as 12.5% of E-mail
from aspammer@website.com is spam.
Der ,,schnelle” braune Fuchs springt
uber den faulen Hund. Le renard brun
¡¡rapide saute par-dessus le chien
paresseux. La volpe marrone rapida
salta sopra il cane pigro. El zorro
marron rpido salta sobre el perro
perezoso. A raposa marrom rpida
salta sobre o 050 preguicoso.
To perform OCR on a scanned PDF document, you could user the following function:

```
import standard.lib.poppler
import standard.C.io
[..]
static PDFText(var pdf\_buffer) {
    var res = "";
    var pdf=PDFLoadBuffer(buffer, "", var err);
    if (pdf) {
        var pages=PDFPageCount(pdf);
        if (pages) {
            for (var i = 0; i < pages; i++) {
                // see if the pdf contains any actual text
                // var page_text = PDFPageText(pdf, i) + "\n;
                // you could also process the attachments
                // echo PDFAttachments(pdf);
                if (PDFPageImage(pdf, 0, "p$i.bmp", "bmp", 3)) {
                    if (!OCR("temp/p0.bmp", var data))
                        res+=data+"\n\n";
                    _unlink("p$i.bmp");
                }
            }
        }
        PDFClose(pdf);
    }
    return res;
}
[..]
    echo PDFText(ReadFile("test.pdf"));
[..]
```

See Concept Framework Documentation for a list containing all the *standard.lib.poppler* PDF read functions.

If you want to process a png or jpeg file, you must first convert it to a uncompressed tiff or bmp. This can be done by using the *win32.graph.freeimage* import library. Note that this is not related to Microsoft Windows. The package name was maintained for backwards-compatibility from the first Concept version. The library itself is cross-platform.

```
import win32.graph.freeimage
[..]
    static ConvertImage(string image_name, string out_name) {
        var type = FreeImage_GetFileType(image_name);
        image_type = ToLower(image_type);
        if (type < 0)
            return true;
        var hBitmap=FreeImage_Load(type, image_name, 0);
        if (hBitmap) {
            // image type should be FIF_BMP or FIF_TIFF
            if (!FreeImage_Save(FIF_BMP, hBitmap, out_name,0)) {
                FreeImage_Unload(hBitmap);
                return false;
            }
            return true;
        }
        return false;
    }
[..]
        if (ConvertImage("input.png", "output.bmp") &&
            (!OCR("output.bmp", var data)))
            echo data;
[..]
```

See the win32.graph.freeimage documentation for a list of all the APIs. Alternatively, you could convert and process images with the more powerful *standard.graph.imagemagick* library based on MagickWand library.

9.14 XML manipulation

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It holds both the data and the structure. Concept framework XML APIs have support for XPath, the XML Path Language, a query language for selecting nodes from an XML document. In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document.

The *XMLDocument* and *XMLNode* classes manage the creation, read and write of the XML documents. Each *XMLDocument* has only one *Root* node that may have multiple children.

The XMLDocument is able to load XML and HTML documents. The HTML documents will be converted to XHTML, to be compatible with its methods and document model.

For all XML processing and serialization Concept Frameworks uses the powerful libxml2 library, ported to Concept via the import library standard.lib.xml. The static function have identical names and parameters with libxml2. You can check the libxml2 documentation if you need low-level access to your XML file.

XMLNode members:

Member	**Return**	**Type**
Name	the node name as a string	property
Handle	the node handle for low-level operations	property
Child	first child as XMLNode or null	property
Parent	parent as XMLNode or null	property
Next	next sibling as XMLNode or null	property
Prev	previous sibling	property
Path	node path as a string	property
Properties	key-value array	property
Content	the node content as string (read/write)	property
CreateNew(name)	a new *name* node	static
AddChild(child)	adds child (as XMLNode)	method
AddNext(child)	adds sibling after (as XMLNode)	method
AddPrev(child)	adds sibling before (as XMLNode)	method
Copy(children=false)	a new cloned node	method

XMLNode must never be created by using the standard constructor. The corect way is:

```
var node = XMLNode.CreateNew("nodeName");
```

Note that a node created with *CreateNew* that won't be added to a document(or another node), must be explicitly freed by using the *Free* method.

XMLDocument provides the following members:

Root : property (XMLNode)
Sets or gets the root node

Encoding : property (string), default is UTF-8
Sets or gets the document encoding

Filename : property (string), default is "document.xml"
Sets or gets the target XML filename

Version : property (string), default is 1.0
Sets or gets the XML version

Errors : property (array)
Gets the parsing errors

ErrorsText : property (string)
Gets the errors as human-readable text

GetXPath (string path)
Evaluate an XPath expression and returns result as an XMLNode array

Create ()
Creates an empty XML document

LoadString (string buffer)
Loads an XML document from the given buffer

Load ()
Loads an XML document from a file specified by the *Filename* property

LoadHTML (string buffer)
Loads an XML document from an HTML string

Save ()
Saves the XML document to a file specified by the *Filename* property

SaveString ()
Saves the XML document to a buffer and returns it.

NewNodeName (string name)
Creates a new XMLNode having the given name. Note that the node is not added to the document.

The following example has the code indented to correspond the XML level.

XMLExample.con

```
include XMLDocument.con

class Main {
    function Main() {
        var doc = new XMLDocument();
        doc.Filename = "sample.xml";
        doc.Create();
        var node = doc.NewNode("XMLTest");
            node.Content = "Hello John!";
            node.SetProperty("Book","Concept Programming");
            var child = doc.NewNode("John");
                child.Content = "John's data";
                var child2 = doc.NewNode("Maria");
                    child2.Content = "Maria's data";
                child.AddNext(child2);
                // dublicate child and add it
                child.AddNext(child.Copy());
            node.AddChild(child);
        doc.Root = node;
        // you could print the content by using
        // var xml = doc.SaveString();
        // save it to disk.
        doc.Save();

        // loading the XML
        var doc2 = new XMLDocument();
        doc2.Filename = "sample.xml";
        doc2.Load();
        // alternatively colud use
        // doc2.LoadString(xml)
        var node2 = doc2.Root;
        echo node2.GetProperty("Book");
            node2 = node2.Child;
            while (node2) {
                echo node2.Name;
                echo "\n";
                node2 = node2.Next;
            }
    }
}
```

The resulting XML file (sample.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLTest Book="Concept Programming">
    <John>John's data</John>
    <John>John's data</John>
    <Maria>Maria's data</Maria>
</XMLTest>
```

When dealing with large XML files, XPath may be a cleaner solution for extracting data. If considering the previous example, the XPath version will be:

```
[..]
var nodes = doc2.GetXPath("/XMLTest[@Book='Concept
    Programming']/John");
echo nodes;
[..]
var len = length nodes;
for (var i=0; i<len; i++) {
    var node = nodes[i];
    echo node.Name+"\n";
}
[..]
```

Will return all the nodes named John under the XMLTest node with a property called Book that is “Concept Programming”, in our case two nodes.

```
Array {
        [0] => XMLNode
        [1] => XMLNode
}
John
John
```

You can also select by node values. Let's consider the following XML snippet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
    <book category="COOKING">
      <title lang="en">Everyday Italian</title>
      <author>Giada De Laurentiis</author>
      <year>2005</year>
      <price>30.00</price>
    </book>
[...]
```

XPath lets you create complex queries, like "/bookstore/book[price>35]/title" selecting all the title nodes with a price higher than 35. All of the GetXPath results are stored in arrays of XMNode.

9.15 XSLT and XSL-FO

XSLT (Extensible Stylesheet Language Transformations) is a language for transforming XML documents into other XML documents, or other objects such as HTML for web pages, plain text or into XSL Formatting Objects which can then be converted to PDF or PostScript, RTF and PNG (when using Apache FOP).

XSL Formatting Objects, or XSL-FO, is a markup language for XML document formatting which is most often used to generate PDFs. XSL-FO is part of XSL (Extensible Stylesheet Language), a set of W3C technologies designed for the transformation and formatting of XML data.

Concept Frameworks has two XSLT processing import libraries: standard.lib.xslt, based on Sablotron and standard.lib.xslt2 based on libxslt2.

standard.lib.xslt2 is recommended because of its extended functionality like the ability to call Concept code directly from the XSLT file.

There are just three static function that manage the entire process:

XSLTRegister (object)
Register a Concept class to be available for execution from XSLT file

XSLTProcess (string xml_data, string xsl_data[, array parameters])
Register a Concept class to be available for execution from XSLT file. The function returns a string containing the transformed data. The

optional *parameters*, if set, must be in the form [“variable1”, “value1”, “variable2”, “value2”...].

XSLTError ()
Returns the errors in *XSLTProcess* or an empty string

XSLT source file can contain complex logic with conditions, recursive templates, loops and Concept function calls. Note that only static function members can be called.

XSLTExample.con

```
import standard.lib.xslt2
import standard.C.io

class Main {
    static function foo(name) {
        return "This text comes from Concept($name)";
    }

    function Main() {
        XSLTRegister(this);
        echo XSLTProcess(ReadFile("test2.xml"),
            ReadFile("test2.xsl"));
        var err = XSLTError();
        if (err)
            echo err;
    }
}
```

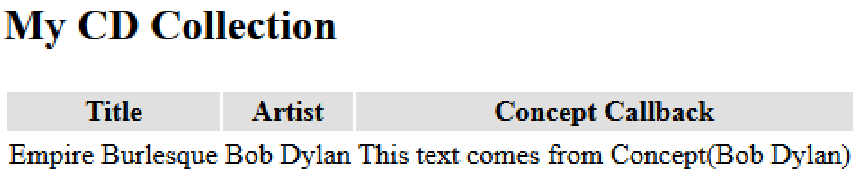
The corresponding XSLT file: **transform.xsl**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:csp="http://www.devronium.com/csp"
    extension-element-prefixes="csp">

<!-- optional: <xsl:output method="html" indent="yes"/> -->

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="0">
    <tr bgcolor="#e0e0e0">
      <th>Title</th>
      <th>Artist</th>
      <th>Concept Callback</th>
    </tr>
    <xsl:for-each select="catalog/cd[artist='Bob Dylan']">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
      <!-- Call a Concept function (Main.foo) -->
      <td><xsl:value-of select="csp:Main.foo(artist)"/></td>
     </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Note that *extension-element-prefixes="csp"* must be added for the Concept callback to work.

data.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
    <cd>
        <title>Empire Burlesque</title>
        <artist>Bob Dylan</artist>
        <country>USA</country>
        <company>Columbia</company>
        <price>10.90</price>
        <year>1985</year>
    </cd>
    <cd>
        <title>Still got the blues</title>
        <artist>Gary Moore</artist>
        <country>UK</country>
        <company>Virgin records</company>
        <price>10.20</price>
        <year>1990</year>
    </cd>
```



My CD Collection

Title	Artist	Concept Callback
Empire Burlesque	Bob Dylan	This text comes from Concept(Bob Dylan)

Figure 9.2:
XSLTExample.con output

```
    <cd>
        <title>For the good times</title>
        <artist>Kenny Rogers</artist>
        <country>UK</country>
        <company>Mucik Master</company>
        <price>8.70</price>
        <year>1995</year>
    </cd>
</catalog>
```

The output:

```
<html>
    <body>
        <h2>My CD Collection</h2>
        <table border="0">
            <tr bgcolor="#e0e0e0">
                <th>Title</th>
                <th>Artist</th>
                <th>Concept Callback</th>
            </tr>
            <tr>
                <td>Empire Burlesque</td>
                <td>Bob Dylan</td>
                <td>This text comes from Concept(Bob Dylan)</td>
            </tr>
        </table>
    </body>
</html>
```

XSLT is useful for separating the design from the code when creating Concept http:// applications.

XSL-FO transformation are done by using the standard.lib.xslfo import library, based on libfo (part of the xmlroff project). It doesn't cover the entire XSL-FO specification, being recommended only for lightweight documents. For complex documents the use of Apache FOP is recommended (and invoked from the comand line via the *system* API).

Like standard.lib.xslt2, standard.lib.xslfo implements three functions that handle all the processing:

FOTransform (string xsl_file, string out_file, format_mode=FO_FLAG_FORMAT_AUTO, base_xsltfile="", compat=false, validation=false, continue_after_error=true)
Transforms the file specified by xsl_file to the out_file as pdf or post script. Returns 0 on success.

FOTransformString (string xsl_string, string out_file, format_mode=FO_FLAG_FORMAT_AUTO, base_xsltfile="", compat=false, validation=false, continue_after_error=true)
Transforms the file specified string to the out_file as pdf or post script. Returns 0 on success.

FOError ()
Returns the last error in *FOTransform/FOTransformString* or an empty string

XSLFOExample.con

```
import standard.lib.xslfo

class Main {
    function Main() {
        //  The buffer version
        //  if (FOTransformString(ReadFile("table.fo"), "out.pdf")) {
        //      echo FOError();
        //  }

        if (FOTransform("example.fo", "example.pdf"))
            echo FOError();
    }
}
```

table.fo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

  <fo:layout-master-set>
    <fo:simple-page-master master-name="A4">
      <fo:region-body />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="A4">
    <fo:flow flow-name="xsl-region-body">
      <fo:block>Hello World!</fo:block>
    </fo:flow>
  </fo:page-sequence>

</fo:root>
```

The resulting PDF will have an A4 page saying "Hello world!".

standard.lib.xslfo provides limited support to XSL-FO. For complex layouts, Apache FOP is recommended because it supports the full specification. After installing FOP, the same application can be rewritten.

XSLFOExampleFOP.con

```
import standard.C.io

class Main {
    function Main() {
        if (system("fop table.fo out.pdf"))
            echo "Error in fop";
    }
}
```

The *system* static function simply executes a command on the server shell.

As a note, try to avoid generating layouts like reports or HTML pages from Concept code, or any other kind of programming language. XSLT and XSL-FO are standard technologies specially created for generating documents. Both of them are great tools for creating dynamic http-based applications and beautiful, flexible and standards-based reports for your applications.

A great crash-course tutorial on both XSLT and XSL-FO can be found on w3schools.com with lots of examples from simple layouts to complex documents.

9.16 Classic web application basics

You can create classic web applications (http-based) in Concept that run via Concept CGI with Apache, nginx and most web servers that support CGI.

Apache Concept CGI settings (in httpd.conf or separate file included by httpd.conf)

```
<IfModule alias_module>
    # Concept Web - autoconfigure for Apache
    ScriptAlias /con/ "[INSTALLDIR]/bin/"
    AddType application/x-httpd-con .con
    # Uncomment next line if you want cgi for all concept extensions
    # AddType application/x-httpd-con .csp
    AddType application/x-httpd-concept .csp
    Action application/x-httpd-con "/con/concept-cgi"
    <Directory "[INSTALLDIR]/bin/">
        AllowOverride None
        Options None
        Order allow,deny
        Allow from all
    </Directory>
</IfModule>
```

[INSTALLDIR] must be replaced with your installation directory. On Windows, usually C:/Program Files/Concept and on linux, unix and bsd /usr/local. Note that on Windows you must replace concept-cgi with concept-cgi.exe.

Alternatively you can use the Concept module and concept cache module. However this is not recommended for developing because its caching systems may affect the debugging process.

```
LoadModule concept_module modules/mod_concept.so
LoadModule concept_cache_module modules/mod_conceptcache.so
```

Web applications based on http:// are a Conpcet feature, not a purpose, and it's recommended that you create them when you can't use the concept:// protocol.

All web applications must include the WebDocument class to gain access to cookies, sessions and get/post variables. Also, it's recommended that you split your application into two parts: the UI template as an XSLT document and the business part of the application that implements all the logic.

Important WebDocument member:

Content : property, string (default "text/html")
Sets or gets the document content type

UseSessions : property, boolean (default true)
Sessions enables you to store variables dependent of a specific user of the application. This property must be set before *BeginDocument* is called.

Status : property, read-only
Gets the document status. It returns DOCUMENT_NOTINIT, DOCUMENT_INIT or DOCUMENT_DONE

Header (string key, string value)
Adds a header to the document, in the form "key: value". This property must be set before *BeginDocument* is called.

PutHeader (string header)
Adds a header to the document. This property must be set before *BeginDocument* is called.

BeginDocument ()
Changes the document status to DOCUMENT_INIT. This can be called just one time per WebDocument.

EndDocument ()
Changes the document status to DOCUMENT_DONE. It is mandatory to call this function for the data to be sent to the user.

DestroySession ()
Deletes the current session, if *UseSessions* is set to true. If set to false, this function throws a ConceptException.

SessionID ()
Returns the current session id as a string, if *UseSessions* is set to true. If set to false, this function throws a ConceptException.

operator << (string)
Outputs data to the webdocument

Print (string)
An alias for <<operator

Beside these members, you will also need some static functions defined in web.service.api for accessing variables.

WebVar (string variable_name[, buffer_size]) returns variable as a string or array (if multiple values)
Returns a web variable, looking in cookies, get and post variables. If buffer_size is set, it will return up to buffer_size bytes

SessionVar (string variable_name) returns session variable value as a string
Returns a session variable

SetSessionVar (string variable_name, string variable_value)
Sets a session variable

SessionTimeout (number seconds)
Sets the session timeout in seconds

ServerVar (string variable_name)
Gets a web server variable, for example "HTTP_USER_AGENT".

CookieVar (string variable_name) returns cookie variable value as a string
Gets a cookie variable by name

SetCookie (string variable_name, string value, number days_expire, string path, string domain, string secure)
Sets a cookie variable

RemoveCookie (string variable_name, string path, string domain, string secure)

GET ()
Returns an array containing all the GET parameters

POST ()
Returns an array containing all the POST parameters

COOKIE ()
Returns an array containing all the cookies

VARS ()
Returns an array containing all the variables, except the server variables

HelloWorldHTTP.con

```
include WebDocument.con

class Main {
    Main() {
        var doc=new WebDocument();
        doc.UseSessions=true;
        doc.BeginDocument();

        var name = GET()["name"];
        if (name)
            doc << "Hello $name form a concept application!";
        else
            doc << "Hello world form a concept application!";

        doc.EndDocument();
    }
}
```

After creating this file in your web serve root, it can be accessed (assuming that the http server runs on th 8080 port):

```
http://localhost:8080/HelloWorldHTTP.con
```

Showing “Hello world form a concept application!” in your web browser. If you add the “name” variable to the request:

```
http://localhost:8080/HelloWorldHTTP.con?name=Eduard
```

You will get “Hello Eduard form a concept application!”.

As said before, is better to avoid generating documents directly from a concept source file. The correct way to do this is to use XSLT templates.

HelloWorldHTTP2.con

```
include WebDocument.con
include Serializable.con
import standard.C.io
import standard.lib.xslt2

class CDCollection {
    var[] cds;
}

class CD {
    var Artist="";
    var Album="";
}

class Main {
    Main() {
        var doc=new WebDocument();
        doc.UseSessions=true;
        doc.BeginDocument();

        var cds = new CDCollection();
        var cd = new CD();
        cd.Artist = "Nirvana";
        cd.Album = "Nevermind";
        cds.cds += cd;

        cd = new CD();
        cd.Artist = "Nirvana";
        cd.Album = "In utero";
        cds.cds += cd;

        var html = XSLTProcess(ToXML(cds), ReadFile("template.xsl"));
        var err = XSLTError();
        if (err)
```

```
            doc << err;
        else
            doc << html;
        doc.EndDocument();
    }
}
```

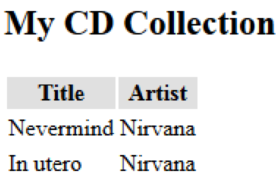
ToXML(cds) will generate:

```
<?xml version="1.0" encoding="UTF-8"?>
<CDCollection>
    <cds type="array">
        <element index="0" type="CD">
            <Artist type="string">Nirvana</Artist>
            <Album type="string">Nevermind</Album>
        </element>
        <element index="1" type="CD">
            <Artist type="string">Nirvana</Artist>
            <Album type="string">In utero</Album>
        </element>
    </cds>
</CDCollection>
```

template.xslt

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:csp="http://www.devronium.com/csp"
    extension-element-prefixes="csp">

    <xsl:output method="html" indent="yes"/>
    <xsl:template match="/">
      <html>
      <body>
      <h2>My CD Collection</h2>
      <table border="0">
        <tr bgcolor="#E0E0E0">
          <th>Title</th>
          <th>Artist</th>
        </tr>
        <xsl:for-each select="CDCollection/cds/element">
        <tr>
```

```
          <td><xsl:value-of select="Album"/></td>
          <td><xsl:value-of select="Artist"/></td>
         </tr>
        </xsl:for-each>
      </table>
      </body>
      </html>
    </xsl:template>
</xsl:stylesheet>
```



My CD Collection

Title	Artist
Nevermind	Nirvana
In utero	Nirvana

Figure 9.3:
HelloWorldHTTP2.con output

HelloWorldHTTP2.con and template.xslt will produce the following HTML source:

```
<html>
    <body>
        <h2>My CD Collection</h2>
        <table border="0">
            <tr bgcolor="#E0E0E0">
                <th>Title</th>
                <th>Artist</th>
            </tr>
            <tr>
                <td>Nevermind</td>
                <td>Nirvana</td>
            </tr>
            <tr>
                <td>In utero</td>
                <td>Nirvana</td>
            </tr>
        </table>
    </body>
</html>
```

Visible in a web browser like figure 9.3.

9.17 Inter-application message exchange

Concept Application Server provides an IPC (inter-process communication) engine enabling applications on the same server to communicate between them, called InterApp.

Each concept:// application has an APID (APplication ID), uniquely identifying the application instance and used by the InterApp subsystem for routing the messages. You can get the APID of the current instance by calling the *GetAPID*() static function.

The messages are sent via the *SendAPMessage*(number target_APID, number message_id, string data). Each message has a number part(message_id) and a string part (data). Messages ids with a value of less than zero are reserved for debugging and framework, and must be avoided.

CApplication has the *ShellDebug* (delegate) property which is called every time an InterApp message is received. The function prototype is:

```
OnInterAppMessage(number SenderAPID, number MSGID, string Data)
```

A message can be sent to all the instances of the same applications, except the sending instance, if the target_APID parameter is set to -1. This is useful for notifying data changes in database applications or, for example, chat or other forms of real-time communication.

Note that this system is available only form concept:// applications. For http:// applications the system can be made available by setting the HandleCGIRequests to 1, APIDHost to localhost, CGTITrustedIP to 127.0.0.1 and CGIPort(default 2663) in concept.ini. The APID 1 is reserved for the Concept Service Manager. The first allocated IP will be 2, except when Concept Services are disabled.

Concept CLI applications cannot have APIDs because are not handled by the Concept Application Server. For console applications, GetAPID() will always return -1.

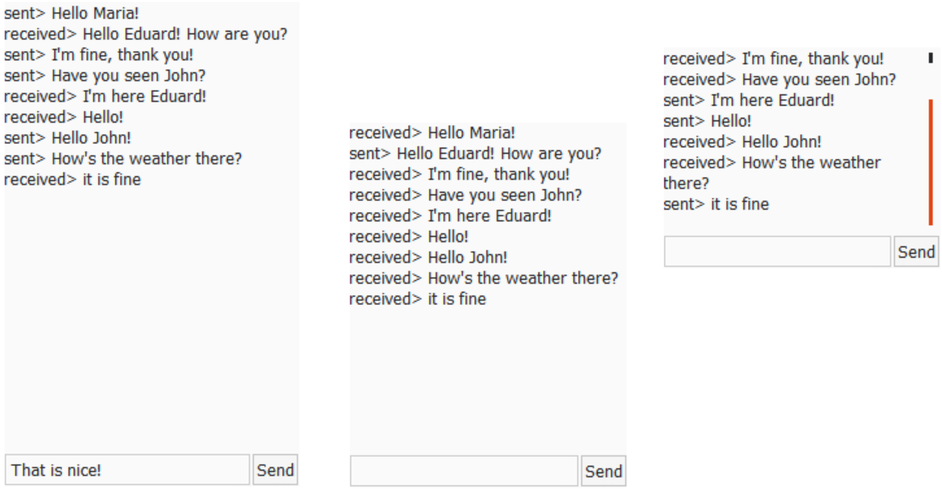
If an application receives a message and doesn't handle it, the message will be lost when the APID data buffer grows over MaxInterAppMessageBuffer(defined in concept.ini). The default MaxInterAppMessageBuffer is 1024 bytes. Messages of size over 1024 bytes won't be guaranteed to be sent. Typically a message should send less than 100-200 bytes, for having a guarantee that will be sent. Also, the Concept Application Server may decide to stop routing messages from a specific application, if flood is suspected or the messages being sent are too large.

As an example, a chat application will be described, using only the InterApp system as means of delivering messages between users.

chat.con

```
include Application.con
include RTextView.con
include REdit.con
include RButton.con
include RVBox.con
include RHBox.con
include RScrolledWindow.con

class MainForm extends RForm {
    var textview;
    var edit;
    var scroll;

    MainForm(owner) {
        super(owner);
        var vbox = new RVBox(this);
        vbox.Show();

        scroll = new RScrolledWindow(vbox);
        scroll.VScrollPolicy = scroll.HScrollPolicy =
            POLICY_AUTOMATIC;
        scroll.Packing = PACK_EXPAND_WIDGET;
        scroll.Show();

        textview = new RTextView(scroll);
        textview.Wrap = WRAP_WORD;
        textview.ReadOnly = true;
        textview.Show();

        var hbox = new RHBox(vbox);
        hbox.Packing = PACK_SHRINK;
        hbox.Show();

        edit = new REdit(hbox);
        edit.Packing = PACK_EXPAND_WIDGET;
        // pressing enter triggers SendTextMessage
        edit.OnActivate = this.SendTextMessage;
        edit.Show();

        var button = new RButton(hbox);
        button.Caption = "Send";
        button.Packing = PACK_SHRINK;
        button.OnClicked = this.SendTextMessage;

        button.Show();
        edit.GrabFocus();
    }

    SendTextMessage(Sender, EventData) {
        // note that edit.Text is copied in a variable
        // for optimizing network traffic (each edit.Text
        // causes a request to the client)
        var text = edit.Text;

        // send to all instances
        SendAPMessage(null, 1, text);
        textview.AddText("sent> $text\n");
        scroll.ScrollDown();

        edit.Text = "";
    }

    OnInterAppMessage(SenderAPID, MSGID, Data) {
        if (MSGID == 1) {
            textview.AddText("received> $Data\n");
            scroll.ScrollDown();
        }
    }
}

class Main {
    Main() {
        try {
            var mainform=new MainForm(null);
            var Application=new CApplication(mainform);
            Application.ShellDebug=mainform.OnInterAppMessage;
            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            CApplication::Message(Exception, "Uncaught exception",
                MESSAGE_ERROR);
        }
    }
}
```

This will enable for all users connected to the same application to send text messages to each other (see figure 9.4).



```
sent> Hello Maria!
received> Hello Eduard! How are you?
sent> I'm fine, thank you!
sent> Have you seen John?
received> I'm here Eduard!
received> Hello!
sent> Hello John!
sent> How's the weather there?
received> it is fine
```

That is nice!

```
received> Hello Maria!
sent> Hello Eduard! How are you?
received> I'm fine, thank you!
received> Have you seen John?
received> I'm here Eduard!
received> Hello!
received> Hello John!
received> How's the weather there?
received> it is fine
```

```
received> I'm fine, thank you!
received> Have you seen John?
sent> I'm here Eduard!
sent> Hello!
received> Hello John!
received> How's the weather
there?
sent> it is fine
```

Figure 9.4:
chat.con output

9.18 String and time functions

standard.lib.str provides a set of easy to use static function for string manipulation. For a complete list of the string static functions, you should check the Concept Framework documentation.

A list with most used string functions:

string StrReplace (string original_string, string what, string replace)
Replaces *what* with *replace* in a copy of textitoriginal_string and returns it

number Pos (string original_string, string what)
Searches for *what* in textitoriginal_string. If its found, it returns the substring index in origianl string + 1. If not found, it returns 0. It uses the Boyer-Moore algorithm to perform a very fast search.

string SubStr (string original_string, number start, number len)
Returns the substring starting the start position and having len characters. If original string end is overpassed, the len parameters gets automatically shrunken internally.

array StrSplit (string original_string, string delimiter, empty_strings=false)
Splits the given original_string by delimiter. If the empty_strings is set to true and two consecutive delimiters are encountered, in the returned array an empty string will be added. Returns an array of sub strings.

array StrNumberSplit (string original_string, string delimiter, empty_strings=false)
Identical with *StrSplit* but returns instead an array of numbers

string chr (number order)
Returns the character identified by *order* in ASCII table.

number ord (string character)
Returns the ASCII value for the given character.

string trim (string)
Trims the given string left and right, removing any leading or trailing spaces, newlines or tabs and returns it.

string ltrim (string)
Trims the given string, removing any leading spaces, newlines or tabs and returns it.

string rtrim (string)
Trims the given string, removing any trailing spaces, newlines or tabs and returns it.

string ToUpper (string)
Returns ASCII string in uppercase

string ToLower (string)
Returns ASCII string in lowercase

string UTF8ToUpper (string)
Returns UTF8 string in uppercase

string UTF8ToLower (string)
Returns UTF8 string in lowercase

number UTF8Length (string)
Returns the length in characters of the UTF8 string.

number calc (string)
Evaluates the arithmetical expression described by string, and returns the result. For example *calc*(”1+2+3*4”) will return 15.

```
import standard.lib.str
[..]
echo StrSplit("This is a test", " ");
[..]
```

Will split the given string by space character and print:

```
Array {
       [0] => This
       [1] => is
       [2] => a
       [3] => test
}
```

Concept also supports regular expressions, both POSIX and Perl. See Concept Framework documentation for *standard.lib.regex* and *standard.lib.preg*.

For example:

```
import standard.lib.regex
[..]
IsEmailValid(email) {
    return regex(email, "{1,64}@{1,255}");
}
[..]
```

Or the Perl-regular expression:

```
import standard.lib.preg
[..]
IsEmailValid(email) {
    return preg(email,
        "[A-Za-z0-9_\\.\\-]+@[A-Za-z0-9_\\.\\-]+\.[A-Za-z0-9_\\.\\-]+");
}
[..]
```

The Concept time functions are just wrappers for the C time functions. All of the time functions are defined in *standard.C.time* import library.

```
import standard.C.time

class Main {
    function Main() {
        var t=time();
        var arr=localtime(t);
        var fmt="%d.%m.%y\n";
        echo strftime(arr, "%d.%m.%y");
    }
}
```

Will print, for example, for Januar 19, 2014: “19.01.2014”. *arr* contains the current time as an array:

```
Array {
        [0,"tm_hour"] => 13
        [1,"tm_isdst"] => 0
        [2,"tm_mday"] => 19
        [3,"tm_min"] => 19
        [4,"tm_mon"] => 0
        [5,"tm_sec"] => 57
        [6,"tm_wday"] => 0
        [7,"tm_yday"] => 18
        [8,"tm_year"] => 114
}
```

See the Concept Framework documentation for a complete list of the APIs.

9.19 Spell check and phonetics

With Concept Framework you can perform spell checks for various languages. The *SpellChecker* class, defined in SpellChecker.con, can provide suggestions for a misspelled word. *SpelChecker* is the high level version of *standard.lib.hunspell* import library, based on Hunspell. Hunspell is a spell checker and morphological analyzer designed for languages with rich morphology and complex word compounding and character encoding, originally designed for the Hungarian language[1]. It supports many languages now, including English, German or French. Hunspell is the spell checker for lots of projects, for example Apple OS X 10.6 or later, Eclipse, Google Chrome, LibreOffice and OpenOffice, Firefox and Thunderbird.

The SpellChecker class implements the following methods:

string CheckSpell (string str, var misspelled_words=null)
: Checks the spelling for *str*, and returns a correction suggestion. If *misspelled_words* is set, it will contain a list of misspelled words, separated by comma.

array Check (string word, null_if_correct=true)
: Checks if the given *word* is correctly spelled. If not, it returns an array of suggested words. If *null_if_correct* is set to false, it will also return suggestions for correctly spelled words.

SpellChecker (language_path)
: The constructor. Language path is the path of the .aff and .dic file. For example, if *language_path* is res/en_GB, you should have two files: res/en_GB.aff and res/en_GB.dic

static string Suggest (string str, var misspelled_words=null)
: Can be called only by a succesful call to *SpellChecker::Static*. Checks the spelling for *str*, and returns a correction suggestion. If *misspelled_words* is set, it will contain a list of misspelled words, separated by comma.

[1] http://en.wikipedia.org/wiki/Hunspell, January 30, 2014

static SpellChecker Static (language_path)

Loads a dictionary (.aff and .dic files), and constructs a static spell checker, enabling the use of *SpellChecker::Suggest* member. Returns the initialized *SpellChecker* object.

Assuming that you downloaded a set of dictionaries, including for English (GB), you could try:

```
include SpellChecker.con

class Main {
   function Main() {
      var spell = new SpellChecker("en_GB");
      echo spell.Check("tes");
   }
}
```

en_GB.aff and *en_GB.dic* must be on the application's root for this example to work. The previous example will output:

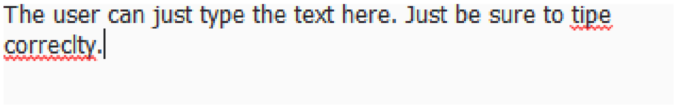
```
Array {
        [0] => set
        [1] => yes
        [2] => tea
        [3] => tees
        [4] => ties
        [5] => teas
        [6] => ates
        [7] => tens
        [8] => test
        [9] => toes
        [10] => est
        [11] => tee
        [12] => ten
        [13] => mes
        [14] => bes
}
```

You could also use the *RTextView* with *SpellChecker*. The following snippet underlines the misspelled words by using. **SpellExample.con**

```
include Application.con
include RTextView.con
include SpellChecker.con

class MyForm extends RForm {
    MyForm(Owner) {
        super(Owner);
        var textview = new RTextView(this);
        textview.Text = "The user can just type the text here";
        textview.Wrap = WRAP_WORD;

        // create the tag
        var error_tag = textview.CreateStyle("SpellError");
        error_tag.Underline = UNDERLINE_ERROR;

        SpellChecker::Static("en_GB");
        textview.OnKeyRelease = this.OnTextEntered;
        textview.Show();
    }

    OnTextEntered(Sender, EventData) {
        var text = Sender.Text;

        SpellChecker::Suggest(text, var ErrWords);
        if (ErrWords)
            Sender.MarkWords(ErrWords, "SpellError");
    }
}

class Main {
    Main() {
        try {
            var Application = new CApplication(new MyForm(null));
            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            echo Exception;
        }
    }
}
```

The output is shown in figure 9.5.

The *standard.lib.languagedetector*, will auto-detect the language the text is



The user can just type the text here. Just be sure to tipe correclty.

Figure 9.5: SpellExample.con output

written in. Based on the language detection result, the specific dictionary could be loaded.

```
string DetectLanguage(string phrase[, var secondary_languages, var
    is_reliable]);
```

The *DetectLanguage* static function will return the most probable language as a string, in which the *phrase* parameter is written in. If *secondary_languages* is set, it will contain an array of strings, containing the detected languages, in order of probabilities. If *is_reliable* is set, it will be set to true, if the language detector is sure about the returned result.

```
import standard.lib.languagedetector

class Main {
    function Main() {
        var lang = DetectLanguage("Do you speak italian?", var
            langs, var is_reliable);
        if (is_reliable)
            echo "Almost sure is $lang\n";
        else
            echo "It may be $lang\n";
        echo langs;
    }
}
```

The output:

```
It may be ENGLISH
Array {
        [0] => ENGLISH
        [1] => ITALIAN
}
```

For some languages, *Soundex*, *Metaphone* and *DoubleMetaphone* may be used for searching or processing misspelled words. These functions are defined in *standard.lib.str* import library.

Soundex is a phonetic algorithm for indexing names by sound, as pronounced in English. The goal is for homophones to be encoded to the same representation so that they can be matched despite minor differences in spelling.

Metaphone is a phonetic algorithm, published by Lawrence Philips in 1990, for indexing words by their English pronunciation. It fundamentally improves on the Soundex algorithm by using information about variations and inconsistencies in English spelling and pronunciation to produce a more accurate encoding, which does a better job of matching words and names which sound similar. As with Soundex, similar sounding words should share the same keys. The Double Metaphone phonetic encoding algorithm is the second generation of this algorithm. It makes a number of fundamental design improvements over the original Metaphone algorithm. It is called "Double" because it can return both a primary and a secondary code for a string; this accounts for some ambiguous cases as well as for multiple variants of surnames with common ancestry. For example, encoding the name "Smith" yields a primary code of SM0 and a secondary code of XMT, while the name "Schmidt" yields a primary code of XMT and a secondary code of SMTboth have XMT in common[2].

Prototypes:

```
string Soundex(string word);
string Metaphone(string word);
string DoubleMetaphone(string word[, var codes]);
```

Each of these function return a phonetic representation of the given word.

```
import standard.lib.str

class Main {
    function Main() {
        echo DoubleMetaphone("metallica");
        echo "\n";
        echo DoubleMetaphone("metalica");
        echo "\n";
```

[2] http://en.wikipedia.org/wiki/Metaphone, on January 30, 2014

```
        echo DoubleMetaphone("mitallik");
        echo "\n";
    }
}
```

The output:

```
MTLK
MTLK
MTLK
```

For every call, for different words, but with identical pronunciation, the same output is generated (MTLK).

9.20 Math

The *standard.C.math* import library maps most of the C's math functions. These are just wrappers to the C functions. The math functions are:

number abs (number x)
Returns the absolute value of x.

number acos (number x)
Returns the principal arc cosine of x, in the interval $[0,\pi]$ radians.

number asin (number x)
Returns the principal arc sine of x, in the interval $[0,\pi]$ radians.

number atan (number x)
Returns the principal arc tangent of x, in the interval $[-\frac{\pi}{2},+\frac{\pi}{2}]$ radians.

number ceil (number x)
Returns the smallest integral value that is not less than x (as a floating-point value).

number cos (number x)
Returns the cosine of x in radians. x is the value representing an angle expressed in radians.

number exp (number x)
Returns the exponential value of *x*.

number fabs (number x)
Returns the absolute value of x ($|x|$).

number floor (number x)
Returns the value of x rounded downward (as a floating-point value).

number fmod (number numer, number denom)
Returns the remainder of dividing *numer*, value of the quotient numerator to *denom*, value of the quotient denominator.

number labs (number x)
Returns the absolute value of parameter x. It is the integer version of *abs.*

number ldexp (number x, number exponent)
The function returns: $x \cdot 2^{exp}$

number log (number x)
Returns the natural logarithm of *x.* If the argument is negative, a domain error occurs.

number log10 (number x)
Returns the common logarithm of *x.* If the argument is negative, a domain error occurs.

number rand ()
Returns a random number, in the interval [0..32767].

number pow (number base, number exponent)
Returns the result of raising *base* to the *power* exponent.

number sin (number x)
Sine of x radians. x is the value representing an angle expressed in radians. One radian is equivalent to $\frac{180}{\pi}$ degrees.

number sqrt (number x)
Returns the square root of x. Note that x must be a positive value.

srand (number seed)
The pseudo-random number generator is initialized using the argument passed as seed.

number tan (number x)
Tangent of x radians.

number round (number x, number decimals)
Rounds x to given *decimals*.

string number_format (number x, number decimals, string decimal_separator, string thousands_separator)
Formats a floating point number to given *decimals* using the given decimal separator and thousand separator, returning the result as a string.

standard.C.math also defines the following constants: M_E (e), M_LOG2E, M_LOG10E, M_LN2, M_LN10, M_PI (π), M_PI_2 ($\frac{\pi}{2}$), M_PI_4 ($\frac{\pi}{4}$), M_1_PI ($\frac{1}{\pi}$), M_2_PI, M_2_SQRTPI, M_SQRT2 ($\sqrt{2}$), M_SQRT1_2 ($\frac{1}{\sqrt{2}}$).

Avoid using *rand* and *srand*, especially in cryptographic functions. For generating pseudo-random numbers, the use of functions implemented in standard.math.rand is recommended.

RandomSeed (number seed)
Initializes the random number generator using the given *seed* (integer).

number RandomInteger (number min, number max)
Returns a random integer between in the interval [*min..max*].

number RandomIntegerX (number min, number max)
Returns a random integer between in the interval [*min..max*]. Similar with *RandomInteger*, but exact. The frequencies of all output values are exactly the same for an infinitely long sequence (Only relevant for extremely long sequences).

number RandomBit ()
Returns a random bit (0 or 1)

number RandomFloat ()
Returns a random floating point number, in the interval [0..1].

For really big numbers, you may use the *standard.math.gmp*, based on the GNU MP library. See the Concept Framework documentation for a complete list of functions.

RandomExmaple.con

```
import standard.math.rand

class Main {
    function Main() {
        // print 10 random integers
        for (var i=0;i<10;i++) {
            echo RandomInteger(0,100);
            echo "\n";
        }
        // print 10 random bits
        for (i=0;i<10;i++) {
            echo RandomBit();
            echo "\n";
        }
        // print 10 random real numbers between 0 and 1
        for (i=0;i<10;i++) {
            echo RandomFloat();
            echo "\n";
        }
        // alternate 10 random integers
        for (i=0;i<10;i++) {
            echo RandomIntegerX(0,100);
            echo "\n";
        }
    }
}
```

Note that the random seed is automatically initialized by the import library.

The math functions are straight forward. The only function that outputs a string is *number_format*.

```
import standard.C.math
[..]
    echo number_format(1123.7891, 2, ".", ",");
[..]
```

Will otuput 1,123.79.

9.21 Serial ports

Some hardware uses serial cables for communicating with computers. For example, low-cost cash registers, medical devices or alarms. The serial ports are easy to use, by simply opening them as special files. The problem arises from compatibility between Windows or Unix-like operating systems. For that, the *standard.io.rs232* provides all the necessary APIs.

OpenComport (number port, number baudrate, number bits=8)
: Opens a COM port for read/write at the given baud rate, using given bits (7 or 8). Returns 0 on success, non-zero on failure. Port 0 is COM1 or /dev/ttyS0, port 1 is COM2 or /dev/ttyS1. A maximum of 16 ports are supported on Windows systems, and 22 on Unix systems.

PollComport (number port, var outputbuffer, number maxsize)
: Read maximum *maxsize* bytes from the COM port into *outputbuffer*. Returns the number of bytes read. This function will block until some data will be available. On error, will return -1.

SendComport (number port, string buffer)
: Sends buffer on the given comport. Returns the number of written bytes, or a negative value on error.

CloseComport (number port)
: Closes the given port.

The example:

```
import standard.io.rs232

class Main {
    function Main() {
        // open COM1 or /dev/ttyS0
        if (!OpenComport(0, 1200)) {
            echo "error opening port";
            return 0;
        }
        // read data from the port
        do {
            var res = PollComport(0, var buf, 100);
            if (res < 0)
                break;
            echo buf;

            // send the data back
            SendComport(0, buf);
        } while (true);
        CloseComport(0);
    }
}
```

Chapter 10

Supported databases

10.1 Database interface

Concept Application Server supports both SQL and NoSQL databases. All SQL-based divers have similar Concept interfaces, for the programmer to be able to switch between them without rewriting the code. Every driver implements 3 classes:

***Connection**
Manages the connection with the database server

***DataSet**
Handles the data returned by a query

***DataRecord**
Used by the dataset to handle column specific data and transformations

The "*" is replaced by a prefix specific to the database system. For example, PostgreSQL uses "PQ", MySQL uses the "My" prefix, SQLite uses "SL", FireBird uses "FB" and ODBC uses "ADO" (short for Abstract Data Objects, not related with Microsoft's ADO).

***Connection** has at least the following members:

DriverOpen (string db, string username, string password, string host, number port, string driver_specific=“”, number flags=0)
Opens a connection to the database server. Returns 0/false if failed.

Close ()
Closes the connection

LastError ()
Returns the last error as a string and sets the error flag to 0 (a second call)

Connection : number property (read-only)
Returns the native connection handle to be used with low-level functions

Most of the *Connection classes supporting transactions, also implement:

StartTransaction ()
Starts a new transaction

EndTransaction (mode)
Ends the current transaction. Mode may be TRANSACTION_ROLLBACK or TRANSACTION_COMMIT

***DataSet** has at least the folowing members:

***DataSet** (*Connection)
Constructor - creates a new dataset object for the given connection

Columns : property (array of strings)
The columns as an array of strings

Connection : property (*Connection)
Gets or sets the *Connection used

CommandText : string property
Sets the query to be executed without bounded parameters

PreparedQuery : string property
Sets the query to be executed with or without bounded parameters

ExecuteNonQuery ()
: Executes a query that produces no results (for example an insert or update). Returns -1 in case of error.

ExecuteQuery ()
: Execute a query that returns a set of data that will be fetched via FetchForward or First/Next. Returns -1 on error.

AddParameter (string parameter_value)
: Adds a parameter value used by *PreparedQuery*.

ClearParameters ()
: Resets the parameters

FieldValues : array of *DataRecord, with column names as keys
: Provides access to resulting columns

FetchForward ()
: Fetches next row and returns true. Returns false if no row is available.

First ()
: Positions the cursor on the first row. Returns false if no data is available. It's recommended that you used *FetchForward* instead(faster).

Next ()
: Fetches next row and returns true. Returns false if no row is available. It's recommended that you used *FetchForward* instead(faster).

Prev ()
: Fetches previous row and returns true. Returns false if no row is available. This is not supported on all servers and NOT recommended. A forward-only cursor is significantly faster and easier on a server.

Last ()
: Fetches the last row and returns true. Returns false if no row is available.

CloseRead (clear_parameters=true)
: Closes the current result set and clears the memory. Has no effect on datasets using *ExecuteNonQuery*.

LastError ()
: Returns the last error as a string and sets the error flag to 0 (a second call)

***DataRecord** has at least the folowing members:

ToNumber ()
: Returns the column data as a number

ToString ()
: Returns the column data as a string, trimming extra spaces for fixed-size columns

ToBuffer ()
: Returns the column data as a string, without trimming extra spaces for fixed-size columns

Each driver can implement its specific methods, however the previous classes and members are common in all the engines.

We will define a basic application template to be used in all the database example, with 2 functions that will be written for each of the supported database.

database_template.con

```
include Application.con
include RForm.con
include RVBox.con
include RTreeView.con
include RScrolledWindow.con
// include specific database classes
// for example, for MySQL:
// include MyDataBases.con

class MyForm extends RForm {
   protected var treeview;
   protected var Connection;

   MyForm(Owner) {
      super(Owner);
```

```
        var vbox = new RVBox(this);
        vbox.Show();

        var edit = new REdit(vbox);
        edit.Text = "SELECT * FROM students";
        edit.Packing = PACK_SHRINK;
        edit.OnActivate = this.OnEnterPressed;
        edit.Show();

        var scroll = new RScrolledWindow(vbox);
        scroll.VScrollPolicy = scroll.HScrollPolicy =
            POLICY_AUTOMATIC;
        scroll.Show();

        treeview = new RTreeView(scroll);
        treeview.Model = MODEL_LISTVIEW;
        treeview.Show();

        this.DoConnect();
    }

    OnEnterPressed(Sender, EventData) {
        // ignore extra spaces
        var query = trim(Sender.Text);
        if (query)
            this.DoQuery(query);
    }

    DoConnect() {
        // specific connect data
    }

    DoQuery(string query) {
        // specific dataset data
    }

    PopulateDatabase() {
        // specific dataset data
    }

    RenderData(dataset) {
        var columns = dataset.Columns;
        var len = length columns;
        treeview.Clear();
        treeview.ClearColumns();
        for (var i = 0; i < len; i++)
            treeview.AddColumn(columns[i]);
        while (dataset.FetchForward()) {
            var column_count = length dataset.FieldValues;
            var item = new [];
            for (var j = 0; j < column_count; j++)
                item[j] = dataset.FieldValues[j].ToString();
            treeview.AddItem(item);
        }
    }

    finalize() {
        // close the database connection
        if (Connection)
            Connection.Close();
    }
}

class Main {
    function Main() {
        try {
            var Application=new CApplication(new MyForm(null));
            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            echo Exception;
        }
    }
}
```

Note that this is not an working example. It is a template to be customized with every database engine. For every driver example, only the *DoConnect*, *PopulateDatabase* and *DoQuery* functions will be described, the rest of the code remaining identical, regardless the used driver.

As a rule, avoid appending string parameters into SQL queries and using them with CommandText/PreparedQuery. This can create a door for an attacker to hijack your queries with unpredictable results. The correct way is to bind parameters by using *DataSet.PreparedQuery* with *DataSet.AddParameter(parameter_value)*.

When working with dates and timestamps, for all database servers the values are written and read as a string using the “YYYY-MM-DD” respectively ”YYYY-MM-DD HH:mm:ss” form, for example ”2014-01-02 15:10:20” will mean January 2nd, 2014, 3:10:20 PM. When performing inserts or selects on dates and timestamps use this format only.

You may use string *strftime(array time, string format), string strftime2(number since_epoch, string format), array strptime(sting time_as_string, string format)* and *number strptime2(sting time_as_string, string format)* defined in standard.C.time import library to convert from and to different formats. See Concept Documentation, topic standard.C.time for more information.

The FieldValue array has keys for each column. You can access a field value by index or by column name. In the following example, indexes are used:

```
while (dataset.FetchForward()) {
    var column_count = length dataset.FieldValues;
    var item = new [];
    for (var j = 0; j < column_count; j++)
        item[j] = dataset.FieldValues[j].ToString();
    treeview.AddItem(item);
}
```

Our example must be query independent (the user enters the query in a text field), but in practice I strongly recommend using field names as keys instead of indexes, because the code is easier to read and debug.

Assuming that the executed query is: “SELECT id, name, registration_date, descrption FROM student”, the recommended way to do it is:

```
while (dataset.FetchForward()) {
    var column_count = length dataset.FieldValues;
    var item = new [];
    item[0] = dataset.FieldValues["id"].ToString();
    item[1] = dataset.FieldValues["name"].ToString();
    item[3] =
        dataset.FieldValues["registration_date"].ToString();
    item[4] = dataset.FieldValues["description"].ToString();
    treeview.AddItem(item);
    }
```

It is important to choose SQL or NoSQL for the right reasons. Each database has its pro and cons, as is the case with SQL and NoSQL.

NoSQL databases are great for document-oriented applications. There they are the stars, but most of them lack transaction support.

All modern databases are relatively fast, so if your query is to slow, avoid blaming on the database server. Most likely your queries are faulty or indexes are not created.

10.2 PostgreSQL

PostgreSQL is an object-relational database system that has the features of traditional proprietary database systems with enhancements to be found in next-generation DBMS systems. PostgreSQL is free and the complete source code is available. PostgreSQL is distributed under a license similar to BSD and MIT. Basically, it allows users to do anything they want with the code, including reselling binaries without the source code. PostgreSQL is a “feature-rich, standards-compliant” database.

PostgreSQL is the recommended database server to use with CAS applications for medium to big data applications.

Every application using the PostgreSQL driver must include PQDataBases.con The database objects prefix is “PQ”, using the following classes:

PQConnection
Manages the connection with the database server

PQDataSet
Handles the data returned by a query

PQDataRecord
Used by the dataset to handle column specific data and transformations

Additionally, it has a specific class called *PQFile* used for managing blobs.

PQFile has the following members:

PQFile (PQConnection con)
Constructor - creates a PQFile object without creating any data in the server

Create (var oid = 0, mode = INV_READ | INV_WRITE)
Creates a new blob/file, setting the oid. Object identifiers (OIDs) are used internally by PostgreSQL as primary keys for various system tables.

Open (number oid, number mode=INV_READ)
Opens a file/blob. Mode can be INV_READ for read-only, INV_WRITE for write-only or a combination of the two.

Seek (number offset, number whence=PGSQL_SEEK_SET)
Moves the read/write cursor to the given position (offset, in bytes), using *whence* as a position flag: PGSQL_SEEK_SET - beginning of the file, PGSQL_SEEK_END - end of file, PGSQL_SEEK_CUR - current position. The file must be opened.

Tell ()
Returns the current cursor position, in bytes, from file start(PGSQL_SEEK_SET). The file must be opened.

Read (number bytes)
Reads a chunk of data from the file (of *bytes* size) and returns it as a string buffer. The file must be opened.

Write (string data)
Writes a chunk of data from to the file. The file must be opened.

Close ()
Closes the PQFile, if opened.

Create a database on the PostgreSQL server called ConceptTestDB. Let's consider the example template described in database interface.

First of all, add the include file at the beginning of the file:

```
include PQDataBases.con
```

The example template *DoConnect* function will use the following code:

```
    DoConnect() {
        Connection = new PQConnection();
        if (!Connection.DriverOpen("ConceptTestDB", "username",
            "password", "localhost")) {
            CApplication.Message("Error connecting to the database");
            Connection = null;
            return;
        }
        // connected !
        // populate the table
        this.PopulateDatabase();
    }
```

Replace *username* and *password* with your database user name and password.

Run this into your favorite PostgreSQL client or in the *psql* command line client, on the ConceptTestDB database.

```
CREATE TABLE students (
    id                integer,
    name              varchar(40),
    registration_date timestamp DEFAULT current_timestamp,
    description       text,
    PRIMARY KEY(id)
);
```

This is called a DDL, short for data definition language.

The *PopulateDatabase* member function:

```
    protected PopulateDatabase() {
        var dataset = new PQDataSet(Connection);
        dataset.CommandText = "START TRANSACTION";
        dataset.ExecuteNonQuery();

        var err = dataset.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }
        // delete all records
        dataset.CommandText = "DELETE FROM students";
        dataset.ExecuteNonQuery();
        err = dataset.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }

        for (var i=0; i < 10 ; i++) {
            var dataset2 = new PQDataSet(Connection);
            dataset2.PreparedQuery = "INSERT INTO students(name,
                registration_date, description) VALUES (:1, :2, :3)";
            // first parameter: name
            dataset2.AddParameter("Student name${i+1}");
            // a date field
            dataset2.AddParameter("2014-01-02 10:00:00");
            dataset2.AddParameter("Here you can add notes for
                name${i+1}");
            dataset2.ExecuteNonQuery();

            err = dataset2.LastError();
            if (err) {
                CApplication.MessageBox(err, "Error");
                return;
            }
        }

        dataset.CommandText = "COMMIT";
        dataset.ExecuteNonQuery();
        err = dataset.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }
    }
```

The example template *DoQuery* function will use the following code:

```
    DoQuery(string query) {
        var dataset = new PQDataSet(Connection);
        dataset.CommandText = query;
        dataset.ExecuteQuery();
        var err = dataset.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }

        this.RenderData(dataset);
    }
```

Note that you must place every INSERT/UPDATE or DELETE statement in a transaction. Only when you call COMMIT the data changes will be actually made available.

PostgreSQL is perfect for any kind of data application. It really shines in concurrent modes, when working with long-lasting transactions. It also scales nicely, when the database becomes relatively big.

For more information about PostgreSQL syntax, indexes, sequence generators, procedures and advanced features, check the documentation available on postgresql.org web page.

10.3 SQLite

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain.

SQLite is great for small to medium CAS applications, due to its light design.

Every application using the SQLite driver must include SLDataBases.con The database objects prefix is "SL", using the following classes:

SLConnection
Manages the connection with the database server

SLDataSet
Handles the data returned by a query

SLDataRecord
Used by the dataset to handle column specific data and transformations

A SQL database can be created by simply opening the database file. A database can be also created by the use of the *sqlite3* command line utility.

```
sqlite3 ConceptTestDB
```

Then you should run the DDL:

```
CREATE TABLE students (
   id                integer PRIMARY KEY AUTOINCREMENT,
   name              varchar(40),
   registration_date datetime,
   description       text
);
```

SQLite doesn't use any username or password for data acces, the only parameter needed for *DriverOpen* is database name.

Let's consider the example template described in database interface.

First of all, add the include file at the beginning of the file:

```
include SLDataBases.con
```

The example template *DoConnect* function will use the following code:

```
   DoConnect() {
      Connection = new SLConnection();
      if (!Connection.DriverOpen("ConceptTestDB")) {
         CApplication.Message("Error connecting to the database");
         Connection = null;
         return;
      }
      // connected !
      // populate the table
      this.PopulateDatabase();
   }
```

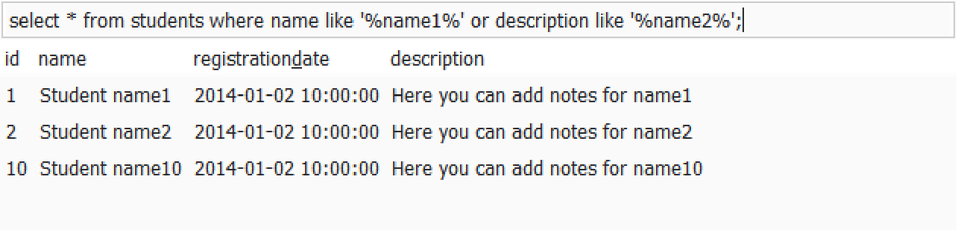
The *PopulateDatabase* member function:

```
protected PopulateDatabase() {
    var dataset = new SLDataSet(Connection);
    dataset.CommandText = "BEGIN TRANSACTION";
    dataset.ExecuteNonQuery();

    var err = dataset.LastError();
    if (err) {
        CApplication.MessageBox(err, "Error");
        return;
    }

    // delete all records
    dataset.CommandText = "DELETE FROM students";
    dataset.ExecuteNonQuery();
    err = dataset.LastError();
    if (err) {
        CApplication.MessageBox(err, "Error");
        return;
    }

    for (var i=0; i < 10 ; i++) {
        var dataset2 = new SLDataSet(Connection);
        dataset2.PreparedQuery = "INSERT INTO students(name,
            registration_date, description) VALUES (?, ?, ?)";
        // first parameter: name
        dataset2.AddParameter("Student name${i+1}");
        // a date field
        dataset2.AddParameter("2014-01-02 10:00:00");
        dataset2.AddParameter("Here you can add notes for
            name${i+1}");
        dataset2.ExecuteNonQuery();

        err = dataset2.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }
    }

    dataset.CommandText = "COMMIT";
    dataset.ExecuteNonQuery();
    err = dataset.LastError();
    if (err) {
        CApplication.MessageBox(err, "Error");
        return;
    }
}
```

The example template *DoQuery* function will use the following code:

```
DoQuery(string query) {
    var dataset = new SLDataSet(Connection);
    dataset.CommandText = query;
    dataset.ExecuteQuery();

    var err = dataset.LastError();
    if (err) {
        CApplication.MessageBox(err, "Error");
        return;
    }

    this.RenderData(dataset);
}
```

Figure 10.1 shows the result.



select * from students where name like '%name1%' or description like '%name2%';

id	name	registrationdate	description
1	Student name1	2014-01-02 10:00:00	Here you can add notes for name1
2	Student name2	2014-01-02 10:00:00	Here you can add notes for name2
10	Student name10	2014-01-02 10:00:00	Here you can add notes for name10

Figure 10.1:
SQLite sample output

Regardless of the database engine, don't forget to create indexes to optimize your WHERE, GROUP BY, and ORDER BY clauses.

Note that by default, SQLite is in auto-commit mode. This means that you don't need to issue a BEGIN TRANSACTION/COMMIT for every INSERT/UPDATE or DELETE statement. However, I recommend you explicitly use start transaction/commit.

The only limit for creating complex data applications using SQLite is the locking mechanism. As of version 3.7 (the current release at the time the book was written), it lacks the support for row-locking. It has however a kind of table-locking, as a result when locking a row, the entire table will be locked. For applications using limited concurrency, or short-lasting transactions, it is the perfect database.

For more information about SQLite syntax, indexes, procedures and advanced features, check the documentation available on sqlite.org web page.

10.4 MySQL

MySQL is (as of July 2013) the world's second most widely used open-source relational database management system (RDBMS), after SQLite. The main pro for using MySQL seems to be its popularity and samples.

I recommend MySQL only for non-transactional or short-lived transaction-based applications. Although it has transaction support in its InnoDB engine, it has some problems when working in high concurrency. It is a great server for classic web applications, but I don't recommend it for using in enterprise applications, except if you are a MySQL guru and know how to deal with it.

Every application using the MySQL driver must include MyDataBases.con The database objects prefix is "My", using the following classes:

MyConnection
Manages the connection with the database server

MyDataSet
Handles the data returned by a query

MyDataRecord
Used by the dataset to handle column specific data and transformations

First of all, add the include file at the beginning of the file:

```
include MyDataBases.con
```

The example template *DoConnect* function will use the following code:

```
    DoConnect() {
        Connection = new MyConnection();
        if (!Connection.DriverOpen("ConceptTestDB", "username",
            "password", "localhost")) {
            CApplication.Message("Error connecting to the database");
            Connection = null;
            return;
        }
        // connected !
        // populate the table
        this.PopulateDatabase();
    }
```

Replace *username* and *password* with your database user name and password.

Run this into your favorite MySQL client or in the *mysql* command line client, on the ConceptTestDB database.

```
CREATE TABLE students (
    id                int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name              varchar(40),
    registration_date datetime,
    description       text
) ENGINE = InnoDB;
```

Note the ENGINE parameter. MySQL uses multiple engines, like MyISAM and InnoDB. MyISAM lacks transactional support, and I recommend the use of InnoDB (the default engine for MySQL).

The *PopulateDatabase* member function:

```
    protected PopulateDatabase() {
        var dataset = new MyDataSet(Connection);
        dataset.CommandText = "START TRANSACTION";
        dataset.ExecuteNonQuery();
        var err = dataset.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }

        // delete all records
        dataset.CommandText = "DELETE FROM students";
        dataset.ExecuteNonQuery();
        err = dataset.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }

        for (var i=0; i < 10 ; i++) {
            var dataset2 = new MyDataSet(Connection);
            dataset2.PreparedQuery = "INSERT INTO students(name,
                registration_date, description) VALUES (?, ?, ?";
            // first parameter: name
            dataset2.AddParameter("Student name${i+1}");
            // a date field
            dataset2.AddParameter("2014-01-02 10:00:00");
            dataset2.AddParameter("Here you can add notes for
                name${i+1}");
            dataset2.ExecuteNonQuery();

            err = dataset2.LastError();
            if (err) {
                CApplication.MessageBox(err, "Error");
                return;
            }
        }

        dataset.CommandText = "COMMIT";
        dataset.ExecuteNonQuery();
        err = dataset.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }
    }
```

The example template *DoQuery* function will use the following code:

```
    DoQuery(string query) {
        var dataset = new MyDataSet(Connection);
        dataset.CommandText = query;
        dataset.ExecuteQuery();

        var err = dataset.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }

        this.RenderData(dataset);
    }
```

Note that by default, MySQL is in auto-commit mode. This means that you don't need to issue a START TRANSACTION/COMMIT for every INSERT/UPDATE or DELETE statement. However, I recommend you explicitly use start transaction/commit.

Is difficult to say that server is the better or fastest. Almost always it ends up to the one best suited for the given task and sometimes, the programmer personality.

The MySQL driver was the first native driver in the Concept Framework, being a mature and stable.

10.5 Firebird

Firebird is an open source SQL relational database management system that runs on Linux, Windows, and a variety of Unix. The database forked from Borland's open source edition of InterBase in 2000, but since Firebird 1.5 the code has been largely rewritten.

It is a feature rich environment, suited for all kinds of applications, from small to complex. It is fun and light weight with fairly good documentation.

Every application using the Firebird driver must include FBDataBases.con The database objects prefix is "FB", using the following classes:

FBConnection
Manages the connection with the database server

FBDataSet
Handles the data returned by a query

FBDataRecord
Used by the dataset to handle column specific data and transformations

Additionally, the FBConnection class implements two additional methods (that are found also in the ODBC driver):

StartTransaction (array options = null)
Starts a new transaction and returns the transaction handle, or -1 on error. options is an array of integer values representing firebird-specific options. Refer to the Firebird's *isc_start_transaction* documentation for a list with all the posible values.

EndTransaction (mode, retain=false, transaction_handle = null)
Ends the transaction with *mode*. Mode can be TRANSACTION_ROLLBACK or TRANSACTION_COMMIT. You can provide an explicit transaction handle. If not set, the EndTransaction will use the last transaction initialized by StartTransaction.

Note that the driver automatically identifies queries like "START TRANSACTION"/"BEGIN TRANSACTION" or "COMMIT" and "ROLLBACK" (not case sensitve) and calls *StartTransaction* respectively, *EndTransaction* automatically. This is done for “duck” (see duck typing) compatibility with the rest of the drivers.

First of all, add the include file at the beginning of the file:

```
include FBDataBases.con
```

The example template *DoConnect* function will use the following code:

```
    DoConnect() {
        Connection = new FBConnection();
        if (!Connection.DriverOpen("/path/to/ConceptTestDB.fdb",
            "username", "password", "localhost")) {
            CApplication.Message("Error connecting to the database");
            Connection = null;
            return;
        }
        // connected !
        // populate the table
        this.PopulateDatabase();
    }
```

Replace *username* and *password* with your Firebird database user name and password. By default, you can use “SYSDBA” as user name and “materkey” as password. Instead of using an absolute path for the database, you could use an alias. Refer to the Firebird documentation for aliases.conf file specification.

Run this into your favorite Firebird client:

```
CREATE TABLE students (
    id                int not null,
    name              varchar(40),
    registration_date datetime,
    description       blob sub_type text,
    constraint pk_students primary key (id)
);
```

The *PopulateDatabase* member function:

```
    protected PopulateDatabase() {
        Connection.StartTransaction();

        var dataset = new FBDataSet(Connection);
        var err = dataset.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }

        // delete all records
        dataset.CommandText = "DELETE FROM students";
        dataset.ExecuteNonQuery();
        err = dataset.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }

        for (var i=0; i < 10 ; i++) {
            var dataset2 = new FBDataSet(Connection);
            dataset2.PreparedQuery = "INSERT INTO students(name,
                registration_date, description) VALUES (?, ?, ?)";
            // first parameter: name
            dataset2.AddParameter("Student name${i+1}");
            // a date field
            dataset2.AddParameter("2014-01-02 10:00:00");
            dataset2.AddParameter("Here you can add notes for
                name${i+1}");
            dataset2.ExecuteNonQuery();

            err = dataset2.LastError();
            if (err) {
                CApplication.MessageBox(err, "Error");
                return;
            }
        }

        Connection.EndTransaction(TRANSACTION_COMMIT);
    }
```

The example template *DoQuery* function will use the following code:

```
    DoQuery(string query) {
        var dataset = new FBDataSet(Connection);
        dataset.CommandText = query;
        dataset.ExecuteQuery();

        var err = dataset.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }

        this.RenderData(dataset);
    }
```

Note that you must place every INSERT/UPDATE or DELETE statement in a transaction. Only when you call COMMIT the data changes will be actually made available.

10.6 ODBC

ODBC (Open Database Connectivity) is a standard programming language middleware API for accessing database management systems (DBMS). The designers of ODBC aimed to make it independent of database systems and operating systems.

Where no native driver is available in the Concept Framework, ODBC is an alternative way for connecting to any modern database server. For example, Concept has no native driver for Microsoft SQL Server and Oracle, but can work with these servers via ODBC. Most modern database servers have ODBC drivers.

The connection process for ODBC drivers is based on a connection string or a DSN (Data Source Name).

ODBC classes uses the "ADO" prefix and are defined in DataBases.con (no prefix).

ADOConnection
Manages the connection with the database server

ADODataSet
Handles the data returned by a query

ADODataRecord
Used by the dataset to handle column specific data and transformations

As a note, the ADOConnection class doesn't have a DriverOpen method. Instead, it uses two alternative methods:

OpenDSN (string dsn, string username="", string password="")
Opens a connection based on a DSN. It returns 0 if the connection failed.

Open (string connection_string)
: Opens a connection based on a connection string. It returns 0 if the connection failed.

The list of connection strings includes (but not limited to):

MSSQL
Driver={SQL Server};Server=localhost;Database=pubs;Uid=;Pwd=;

Access
Driver={Microsoft Access Driver (*.mdb)};Dbq=mydatabase.mdb;Uid=;Pwd=;

Oracle
Driver={Microsoft ODBC for Oracle};Server=OracleServer.world;Uid=;Pwd=;

MySQL
Driver={MySQL ODBC 3.51 Driver};SERVER=data.domain.com;PORT=3306; DATABASE=myDatabase;USER=myUsername; PASSWORD=myPassword;OPTION=3;

Firebird
Driver=Firebird/InterBase(r) driver;UID=SYSDBA; PWD=masterkey;DBNAME=/path/to/database.fdb

Interbase
Driver={Easysoft IB6 ODBC};Server=localhost; Database=localhost:mydatabase.gdb;Uid=username;Pwd=password;

IBM DB2
Driver={IBM DB2 ODBC DRIVER};Database=myDbName; hostname=myServerName;port=myPortNum;protocol=TCPIP; uid=myUserName; pwd=myPwd;

Sysbase 12
Driver={SYBASE ASE ODBC Driver};Srvr=localhost; Uid=username;Pwd=password

Informix

Dsn=”;Driver={INFORMIX 3.30 32 BIT};Host=hostname; Server=myserver;Service=service-name;Protocol=olsoctcp;Database=mydb;UID=username;PWD=myPwd;

Mimer SQL

Driver={MIMER};Database=mydb;Uid=myuser;Pwd=mypw;

Paradox 7.x

Provider=MSDASQL.1;Persist Security Info=False; Mode=Read;Extended Properties=’DSN=Paradox;DBQ=/path/to/myDb; DefaultDir=/path/to/myDb;DriverId=538;FIL=Paradox 7.X; MaxBufferSize=2048;PageTimeout=600;’;Initial Catalog=/path/to/mydb

Excel

Driver={Microsoft Excel Driver (*.xls)};DriverId=790;Dbq=/path/to/MyExcel.xls;DefaultDir=/path/to/;

Text

Driver={Microsoft Text Driver (*.txt; *.csv)};Dbq=/path/to/txtFilesFolder/;Extensions=asc,csv,tab,txt;

DBF/FoxPro

Driver={Microsoft dBASE Driver (*.dbf)};DriverID=277;Dbq=/path/to/mydb;

Visual FoxPRO(.DBC)

Driver={Microsoft Visual FoxPro Driver};SourceType=DBC;SourceDB=/path/to/myvfpdb.dbc; Exclusive=No;NULL=NO;Collate=Machine; BACKGROUNDFETCH=NO;DELETED=NO

Visual FoxPRO(Free table dir)

Driver={Microsoft Visual FoxPro Driver};SourceType=DBF; SourceDB=/path/to/myvfpdbfolder;Exclusive=No;Collate=Machine; NULL=NO;DELETED=NO;BACKGROUNDFETCH=NO;

Pervasive

Driver={Pervasive ODBC Client Interface};ServerName=srvname;dbq=@dbname;

Some of the driver are available only on the windows version, depending on the specific ODBC driver cross-platform abilities. It is recommended to avoid using non-portable drivers. In practice, this will be only used when importing data from old/deprecated systems.

Note that the used ODBC driver must be installed on the system hosting Concept Application Server.

Additionally, the ADOConnection class implements two additional methods:

StartTransaction ()
  Starts a new transaction

EndTransaction (mode)
  Ends the transaction with *mode*. Mode can be TRANSACTION_ROLLBACK or TRANSACTION_COMMIT

For example, a MySQL connection, using the ODBC driver, can be made via the MyODBC driver (either 3.51 or 5).

First of all, add the include file at the beginning of the file:

```
include ADODataBases.con
```

The example template *DoConnect* function will use the following code:

```
    DoConnect() {
        Connection = new ADOConnection();
        if (!Connection.OpenString("Driver={MySQL ODBC 3.51
            Driver};SERVER=localhost;PORT=3306;DATABASE=ConceptDBTest;
            USER=username;PASSWORD=password;OPTION=3;")) {
            CApplication.Message("Error connecting to the database");
            Connection = null;
            return;
        }
        this.PopulateDatabase();
    }
```

The *PopulateDatabase* member function:

```
protected PopulateDatabase() {
    Connection.StartTransaction();

    var dataset = new ADODataSet(Connection);

    // delete all records
    dataset.CommandText = "DELETE FROM students";
    dataset.ExecuteNonQuery();
    err = dataset.LastError();
    if (err) {
        CApplication.MessageBox(err, "Error");
        return;
    }

    for (var i=0; i < 10 ; i++) {
        var dataset2 = new ADODataSet(Connection);
        dataset2.PreparedQuery = "INSERT INTO students(name,
            registration_date, description) VALUES (?, ?, ?";
        // first parameter: name
        dataset2.AddParameter("Student name${i+1}");
        // a date field
        dataset2.AddParameter("2014-01-02 10:00:00");
        dataset2.AddParameter("Here you can add notes for
            name${i+1}");
        dataset2.ExecuteNonQuery();

        err = dataset2.LastError();
        if (err) {
            CApplication.MessageBox(err, "Error");
            return;
        }
    }

    Connection.EndTransaction(TRANSACTION_COMMIT);
}
```

The ODBC drivers should be used only when no native driver is available. In practice it is used mostly for data imports from various databases.

10.7 NuoDB

NuoDB is a SQL/ACID compliant distributed database management system. Different from traditional shared-disk or shared-nothing architectures, NuoDB is a new distributed, peer-to-peer, asynchronous approach. It has a distributed object architecture that works in the cloud, which means that when a new server is added in order to scale-up the database, the database runs faster. The database scales out without sharding. The database distributes tasks amongst several processors to avoid bottlenecks of data. It uses peer-to-peer messaging to route tasks to nodes, and it is ACID compliant. In short words, NouDB is a NoSQL-style database system that knows SQL.

NuoDB is the only closed source database system supported natively by the Concept Application Server.

Every application using the NouDB driver must include NuoDataBases.con The database objects prefix is "Nuo", using the following classes:

NuoConnection
Manages the connection with the database server

NuoDataSet
Handles the data returned by a query

NuoDataRecord
Used by the dataset to handle column specific data and transformations

The DriverOpen function has the following prototype:

```
DriverOpen(string db, string username="", string password="",
    string host="localhost", string schema="hello", create=true)
```

Also, NuoConnection adds two properties

```
number property IsolationLevel;
boolean property AutoCommit;
```

Transactions are managed via:

StartTransaction ()
: Starts a new transaction

EndTransaction (mode)
: Ends the transaction with *mode*. Mode can be TRANSACTION_ROLLBACK or TRANSACTION_COMMIT

The code is identical with the Firebird version, with the only difference that the "Nuo" prefix will be used of all objects.

10.8 MongoDB

MongoDB (from "humongous") is a cross-platform document-oriented database system. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster.

MongoDB is able to store native Concept objects, as documents. Instead of tables (having the same columns for every record), MongoDB uses collections. Items in a collection can may different fields.

MongoConnection
: Manages the connection with the database server

MongoDataSet
: Handles the data returned by a query and creates a cursor for navigating between documents

MongoCursor
: Used by the dataset to handle column specific data and transformations

MongoConnection manages the connection with the MongoDB server. The most used members are:

DriverOpen (db=””, user=””, password=””, host=”127.0.0.1”, port=27017)
connects to MongoDB

SetTimeout (number seconds)
sets the operation timeout in seconds

Error ()
returns the connection error as a string

LastError ()
returns last query error

AddUser (string username, string password)
Adds a new user

DropDb (string dbname)
Drops a database

DropCollection (string dbname, string collection, var out=null)
Drops a collection

Eval (string javascript_query, keep_object_types=true)
Evaluates the javascript_query string and returns the result

SimpleIndex (string field, flags=0, var out=null)
Ensures that an index for the given field exists.

Index (array fields, flags=0, var out=null)
Ensures that an index for the given fields exists.

Close ()
Closes the current connection

Every MongoDB query is defiend as an array. Assuming that we have a collection of Test objects, defined by:

```
class Test {
   var Name="Eduard";
   var Age=30;
}
```

The query for returning all Test documents that have *Name* validating "/.*eduaD*/i" or Age = 30, sorted by Name descending, and by Age descending will be:

```
['$query': ['$or': [["Name": "/.*eduaD*/i"], ["Age": 30]]],
    '$orderby': ["Name": -1, "Age": -1]]
```

MongoDB uses reserved words, prefixed by "$". See MongoDB documentation for better understanding these special keys.

The queries are executed via *MongoDataSet*. Notable members of MongoDataSet are:

MongoDataSet (MongoConnection connection, string collection="")
Constructor - creates a new MongoDataSet for the given collection

Query : array property
Holds the MongoDB query

MapReduce (string map, string reduce, skip=0, limit=0, resultCollection="", var out=null, keep_object_types=true)
Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. See MongoDB documentation for more information.

Insert (data, keep_object_types=true, id_field="_id")
Insert *data* as a document into collection, returning the id as a string. Data must be an object or an array.

Count ()
Counts documents from the collection matching *Query*

Remove ()
Removes documents from the collection matching *Query*

Update (object data, update_all=true, keep_object_types=true, force_objects=true)
Updates the data object

Find (fields=null, skip=0, limit=0)
Finds all the objects matching *Query* and returns a *MongoCursor*.

FindOne (fields=null, keep_object_types=true)
Finds and returns the first object matching *Query*. fields may be an array specifying the *fields* to be returned. If *fields* is null, all the document properties will be set.

The concept driver will automatically add a “classof()” member to every documents, if the keep_object_types is set to true, identifying the object type. This is useful when retrieving the data, for the driver to reconstruct a clone of the original object.

The result sets are managed by cursor, handled by the *MongoCursor* class. Members of MongoCursor:

FieldValues : array property
keeps the document values

Next (keep_object_types=true)
Fetches the next object in the cursor

FetchForward ()
An alias to *Next*()

Close ()
Closes the cursor

MongoExample.con

```
#!/usr/local/bin/concept
include MongoDataBase.con

class Test {
    var Name="Eduard";
    var Age=30;
    var[] data;
}

class Main {
    function Main() {
        try {
            var mongo=new MongoConnection();
            if (mongo.Connect("tutorial")) {
                var[] arr;
                for (var i=0;i<10;i++) {
                    var t=new Test();
                    t.Age=i;
                    t.Name+=" "+i;
                    t.data=[1,2,3,4,5,"arraytest",[6,7,8],10];
                    arr[i]=t;
                }

                var dataset=new MongoDataSet(mongo,"people");
                dataset.Insert(arr);

                dataset.Query=['Age': 9];
                var res=dataset.Find();
                if (res) {
                    while (res.Next()) {
                        echo res.FieldValues.Name+":\n";
                        echo res.FieldValues.data;
                    }
                }
                echo "\nElements: "+dataset.Count()+"\n";
            }
            mongo.Close();
        } catch (var exc) {
            echo exc;
        }
    }
}
```

Outputs:

```
Eduard 9:
Array {
        [0,"0"] => 1
        [1,"1"] => 2
        [2,"2"] => 3
        [3,"3"] => 4
        [4,"4"] => 5
        [5,"5"] => arraytest
        [6,"6"] =>
                Array {
                        [0,"0"] => 6
                        [1,"1"] => 7
                        [2,"2"] => 8
                }
        [7,"7"] => 10
}
Elements: 1
```

Remember to create indexes by using *MongoConnection.Index* or *MongoConnection.SimpleIndex* before you use the *$orderby* flags.

MongoDB is a great server when used wisely. MongoDB is great for document-based data application, but don't forget that it lacks transactional support. For application with fixed data, could be great, but for intense writes and concurrent updates will not be safe, unless additional code will be added in the middle-ware, controlling the updates. As a conclusion, don't choose MongoDB because is fast, but choose it because is right for your problem.

10.9 dBase files

The dBase .dbf files are still used in a variety of (old) systems. Concept Frameworks offers basic support for reading and writing dbf files, using the *standard.db.dbase import* library. The support is limited, because dbf files are not meant to be used as a base for a newly created application. The API's are there for helping the data import and export from and to old systems.

ReadDBF.con

```
#!/usr/local/bin/concept
import standard.db.dbase

class Main {
    function GetDB(string name) {
        var handle=DBFOpen(name, false);
        if (!handle)
            throw "Invalid DBF file";

        var info=DBFInfo(handle);
        var fields=info["fieldcount"];
        var count=info["recordcount"];
        var[] res;
        var[] field_names;
        for (var j=0;j<fields;j++) {
            var field_name=DBFFieldInfo(handle, j)["name"];
            field_names[field_name]=field_name;
        }

        for (var i=0;i<count;i++) {
            DBFMove(handle, i);
            var line=new [];
            res[i]=line;
            for (j=0;j<fields;j++) {
                line[field_names[j]]=DBFGet(handle, j);
            }
        }
        var err=DBFLastErrorString(handle);
        if (err)
            throw err;

        DBFClose(handle);
        return res;
    }

    function Main() {
        echo GetDB("TEST.DBF");
    }
}
```

The GetDB function will return a matrix containing all the records in the .dbf. Every line in the matrix will represent one row from the .dbf file. Every row array element will have as key the column name.

DBF function prototypes:

dbhandle DBFOpen (string dbfname, boolean editable=false,
char_conversion=ENUM_dbf_charconv_oem_host,
table_name=dbfname)
Opens a dbf file

DBFClose (dbhandle)
Closes the dbf file

bool DBFMove (dbhandle, number rowindex)

Moves the cursor in the dbf to the give rowindex. Returns true on success, false if no more rows are available.

bool DBFDelete (dbhandle, number rowindex)
Deletes the row on the given position (0 is first). Returns true on success.

bool DBFIsDeleted (dbhandle, number rowindex)
Checks if the row on the given index is deleted. Returns true on success.

bool DBFAddRecord (dbhandle)
Adds a new record at the end of the dbf. Returns true on success.

bool DBFInsertRecord (dbhandle, number rowindex)
Inserts a new record at the on the given row index. Returns true on success.

number DBFFindField (dbhandle, string fieldname)
Returns the index of the given field, or -1 if not found

number DBFLastError (dbhandle)
Returns the last error code

string DBFLastErrorString (dbhandle)
Returns the last error as a human-readable string

array DBFInfo (dbhandle)
Returns the dbf meta info as a key-value array. The keys are: version, flags, fieldcount, recordcount, lastupdate, flags, memo, editable, modified, tablename and format.

array DBFFieldInfo (dbhandle, number column_index)
Returns the field meta info as a key-value array. The keys are: name, type, length and decimals.

string DBFGet (dbhandle, number column_index)
Gets the value for the current row and the given column index as a string.

string DBFGetName (dbhandle, string column_name)
Gets the value for the current row and the given column name as a string.

bool DBFUpdate (dbhandle, number column_index, string new_value)
Updates the column_index for the current row to *new_value*. Returns true on success.

bool DBFUpdateName (dbhandle, string column_name, string new_value)
Updates the column_name for the current row to *new_value*. Returns true on success.

bool DBFNull (dbhandle, number column_index)
Returns true if the column on the given index is null.

bool DBFNullName (dbhandle, string column_name)
Returns true if the column name is null.

The add operation is relatively simple:

```
var handle=DBFOpen(name, true);
DBFAddRecord(handle);
DBFUpdateName(handle, "FIELD_NAME", "Value");
DBFUpdateName(handle, "VALUE", "1.00");
DBFClose(handle);
```

Avoid basing your application on dbf files, being a deprecated system. If light-weight is a must, SQLite should be a better solution.

10.10 Memcached

Memcached is a general-purpose distributed memory caching system. The system uses a clientserver architecture. The servers maintain a keyvalue associative array; the clients populate this array and query it. Keys are up to 250 bytes long and values can be at most 1 megabyte in size. It's not actually a database, is a very fast key-value store.

When dealing with large datasets, only a fraction of the data will be frequently requested by the users. Memcached enables the application to cache the data, avoiding the same query to be sent twice in a short interval to the database server. When a user requests some complex data, the database server will generate it, consuming CPU power and returning the data to the user. When using memcached, when the second user requests the same data before the cache expiring, the database server won't be queried.

The memcached client is managed by the MemCached class, defined in MemCached.con.

Notable members:

AddServer (string host="localhost", number port=11211)
Adds a server to the server list

Timeout
Cache timeout, in seconds. When set to 0 (default), it will never expire.

Add (string key, string data, timeout=-1, flags = 0)
Adds the value for the given key

Set (string key, string data, timeout=-1, flags = 0)
Sets the value for the given key

Get (string key)
Returns the value for the given key

operator[ ](string key)
An alias for *Get(key)*.

Delete (string key, timeout=-1)
Deletes the given key

SetByKey (string masterkey, string key, string data, timeout=-1, flags=0)
Sets the value for the given key and masterkey.

GetByKey (string masterkey, string key)
Returns the value for the given key and masterkey.

DeleteByKey (string masterkey, string key, timeout=-1)
Deletes the given key for the give master key

MemCachedExample.con

```
#!/usr/local/bin/concept
include MemCached.con
include Serializable.con
import standard.lib.cripto

class Student {
    var Name = "Eduard";
    var Notes = "Some notes";
}

class Main {
    var mem;

    CacheCheck(key) {
        // using md5 hashes instead of keys, for caching
        // complex SQL queries (> 240 characters in length)
        var hash = md5(key);
        try {
            var xml = mem[hash];
            if (xml)
                return UnSerialize::Unserialize(xml, true);
        } catch (var exc) {
            // non-existing key
            return null;
        }
    }

    ExecuteSQL(query) {
        // We will assume that this function executed
        // the given query and returned some data
        var student = new Student();
        // cache the student data as xml
        mem.Set(md5(query), student.Serialize(""));
        return student;
    }

    Query(string query) {
        var val = CacheCheck(query);
        if (!val) {
            val = ExecuteSQL(query);
            echo "Returned from the SQL server\n";
        } else
            echo "Returned from Memcached\n";
        return val;
    }

    Main() {
        mem = new MemCached();
        // one hour cache time
        mem.Timeout = 3600;
        mem.AddServer();

        echo Query("select * from some_table where id=10");
    }
}
```

On the first run, the program will output:

```
Returned from the SQL server
Student
```

On the second run, the program will output:

```
Returned from Memcached
Student
```

It is not mandatory to use md5 as a key. You could use plain-text strings, but in our case, a query can have more than 240 characters (the key limit for memcached).

For a complete list of members for MemCached, check the Concept Framework documentation.

10.11 Natural searches with Xapian

Xapian is a highly adaptable toolkit which allows developers to easily add advanced indexing and search facilities to their own applications. It supports the Probabilistic Information Retrieval model and also supports a rich set of boolean query operators.

Applications are only as good as their search is. Xapian allows the user to search for data using natural language. It uses Okapi BM25, a ranking function used by search engines to rank matching documents according to their relevance to a given search query.

The Concept Xapian APIs are using the same classes and prototypes as the standard Xapian library. The Xapian classes are defined in Xapian.con.

The Xapian example has two command line programs: an indexer, a program that will index the data, and a searcher.

DataIndexer.con

```
#!/usr/local/bin/concept
include Xapian.con
import standard.lang.cli
import standard.C.io
import standard.lib.str

class Main {
    function Main() {
        try {
            var arg=CLArg();
            if (length arg != 1) {
                echo "Usage: DataIndexer.con inputfile";
                return -1;
            }

            // Open the database for update, creating a new database
                if necessary.
            var db=new XapianWritableDatabase("XapianTest",
                DB_CREATE_OR_OPEN);

            var indexer=new XapianTermGenerator();
            var stemmer=new XapianStem("english");
            indexer.set_stemmer(stemmer);

            var content=ReadFile(arg[0]);
            var doc=new XapianDocument();
            doc.set_data(content);
            indexer.set_document(doc);
            indexer.index_text(content);
            db.add_document(doc);
        } catch (var exc) {
            echo exc;
        }
    }
}
```

The search will use the database created by the indexer to perform the full text search:

SearchData.con

```
#!/usr/local/bin/concept
include Xapian.con
import standard.lang.cli
import standard.C.io
import standard.lib.str

class Main {
    function Main() {
        try {
            var arg=CLArg();
            if (length arg != 1) {
                echo "Usage: SearchData.con query";
                return -1;
            }

            var db=new XapianDatabase("XapianTest");

            var enquire=new XapianEnquire(db);
            var qp=new XapianQueryParser();

            var stemmer=new XapianStem("english");
            qp.set_database(db);
            qp.set_stemming_strategy(STEM_SOME);
            var query=qp.parse_query(arg[0]);
            echo "Parsed query is: "+query.get_description() +"\n";
            echo "Do you mean
                ’"+db.get_spelling_suggestion(arg[1],2)+"’ ?\n";

            enquire.set_query(query);
            var matches=enquire.get_mset(0,10);

            echo "" + matches.get_matches_estimated() + " result(s)
                found.\n";
            echo "Matches 1-" + matches.size() + ":\n";

            for (var i=matches.begin(); i!=matches.end(); i++) {
                echo ""+ (i.get_rank() + 1) + ": " + i.get_percent()
```

```
                    + "% docid=" + i.get_value();
                echo " ["+i.get_document().get_data()+"]\n\n";
            }

        } catch (var exc) {
            echo exc;
        }
    }
}
```

After running:

```
DataIndexer.con SnowWhite.txt
DataIndexer.con SleepingBeauty.txt
```

The XapianTest database contains the two files. Then, we can query the database:

```
SearchData.con "A tale with a queen and a mirror"
```

The output:

```
Parsed query is: Xapian::Query((a:(pos=1) OR tale:(pos=2) OR
    with:(pos=3) OR a:(pos=4) OR queen:(pos=5) OR and:(pos=6) OR
    a:(pos=7) OR mirror:(pos=8)))
Do you mean ’’ ?
2 result(s) found.
Matches 1-2:
1: 87% docid=1 [At the beginning of the story, a queen sits sewing
    at ..]
2: 48% docid=2 [At the christening of a king and queen’s
    long-wished-for ..]
```

The first result refers to Snow White, the second one, Sleeping Beauty.

If the search query is changed to:

```
SearchData.con "A tale with a princess and a wicked fairy"
```

The output:

```
Parsed query is: Xapian::Query((a:(pos=1) OR tale:(pos=2) OR
    with:(pos=3) OR a:(pos=4) OR princess:(pos=5) OR and:(pos=6) OR
    a:(pos=7) OR wicked:(pos=8) OR fairy:(pos=9)))
Do you mean ’’ ?
2 result(s) found.
Matches 1-2:
1: 88% docid=2 [At the christening of a king and queen’s
    long-wished-for ..]
2: 35% docid=1 [At the beginning of the story, a queen sits sewing
    at ..]
```

Note the reversed order of the results. The first result refers to Sleeping Beauty, the second one Snow White.

Xapian performs extremely fast searches, using natural language. The results in the above example are sorted by relevance.

This kind of full text search is not possible via simple SQL queries. Using SQL databases with Memcached and Xapin can have a positive impact on the usability and user experience.

For a complete reference to the Xapin classes, check Concept Framework documentation and visit xapian.org website.

Chapter 11

Sockets and networking

Network sockets are endpoints of inter-process communication flows across networks, for example the Internet.

Concept supports 4 types of sockets: TCP/IP, UDP, Unix sockets and Bluetooth sockets. Note that Unix sockets are emulated using named pipes on Windows. In addition to plain sockets, Concept Framework also supports SSLv2, v3 and TLS sockets.

The socket low-level APIs are defined in *standard.net.socket* import library. Concept Framework supports both IPv4 and IPv6, on the same APIs. No changes are necessary to the code for the use of IPv6.

The next section will describe only high-level interfaces. For the low-level APIs, check the Concept Framework documentation, standard.lib.socket in the static library section.

11.1 TCP/IP sockets

TCP, short for Transmission Control Protocol, provides reliable, ordered, error-checked delivery of a stream of octets between programs running on computers connected to a local area network, intranet or the public Internet. It basically guarantees that the data was received by the host, or the connection is lost.

For establishing a TCP connection, a client socket must connect to a server listening and accepting connections on a given port. The entire process is handled by the *TCPSocket* class defined in TCPSocket.con.

TCPSocket members:

TCPSocket (number sockdescriptor=-1, number is_ipv6=false)
Constructor, initializes a *TCPSocket*. *sockdescriptor* is the socket descriptor (-1 if this is a new socket). For IPv6 sockets, the *is_ipv6* flag must be set to **true**.

Connect (string host, number port)
Used in client sockets, connects the server at host:port. Returns true if succeeded, false if failed.

Listen (number port, number maxconnections = 0xFF, interface="")
Used in server sockets, listens on the given port and interface. If interface is not set, will listen on all available interfaces. *maxconnections* specifies the maximum concurrent connections. Returns 0 if succeeded.

Accept (return_static_socket=false)
Waits for a connection on a server socket. Returns a TCPSocket object if return_static_socket is false, or a socket file descriptor as a number otherwise.

Close ()
Closes the socket

Read (max_size=0xFFFF)
Reads at most max_size bytes and returns the read buffer. In case of error, it throws an error string.

Write (string buffer)
Sends buffer on the socket. Returns the number of bytes sent. In case of error, it throws an error string.

SetOption (number level, number option, number val)
Sets an option for socket. *level* can be SOL_SOCKET, IPPROTO_TCP. *option* is a socket option [1] and *val* the new value. Returns 0 on success, -1 on error.

[1] Available socket options are (cross-platform): SO_DEBUG, SO_ACCEPTCONN,

GetOption (number level, number option, var val)
Gets an option for socket. *level* can be SOL_SOCKET, IPPROTO_TCP. *option* is a socket option (see bellow). Returns 0 on success and sets the val to the value, -1 on error.

Ipv6 : boolean read-only property
Returns true if socket uses IPv6.

HasData : boolean property
Returns true if data can be read, or a connection event occurred.

Info : array property
Returns an array containing the IP and the port used by the current socket. If the socket is invalid, it will return an empty array.

Socket : number property
Returns the socket file descriptor for use with the low-level APIs

Error : number property
Returns the error code of the last socket operation

Every client socket must first make a successfully call to *Connect* in order to perform *Read* or *Write* operations.

For the socket server you cannot call *Read* or *Write* directly. The function call order is: *Listen*, client_socket = *Accept*(), client_socket.*Read* and/or client_socket.*Write*. A call to Listen will fail, if another server program running on the same machine, listens on the same port(TCP). Note that two different programs can use the same port if the use different protocols (one UDP and another TCP).

Note that there is no guarantee that a call to *Write* will send the entire data buffer. It is important to analyze the result of the Write (the amount of bytes written), and then call Write again for the remaining buffer. This may happen also with *Read*, that may return less that the *max_size*, and a second call to *Read* may be needed. Due to socket send/receive buffer size,

SO_REUSEADDR, SO_KEEPALIVE, SO_DONTROUTE, SO_BROADCAST, SO_LINGER, SO_OOBINLINE, SO_SNDBUF, SO_RCVBUF, SO_SNDLOWAT, SO_RCVLOWAT, SO_SNDTIMEO, SO_RCVTIMEO, SO_ERROR, SO_TYPE, TCP_NODELAY, IP_TOS and IP_TTL. Check the standard C socket documentation for finding out what every options does. In practice, you will rarely use them.

a big buffer written on the socket, may be actually split in multiple network packets.

A minimal TCP server:

```
include TCPSocket.con

class Main {
    Main() {
        var t = new TCPSocket();
        if (t.Listen(2000)) {
            echo "Error in listen\n";
            return -1;
        }


        while (true) {
            try {
                var client=t.Accept();
                if (!client)
                    break;

                echo client.Read();
                client.Write("Hello client!");
                client.Close();
            } catch (var exc) {
                echo exc;
                break;
            }
        }
        t.Close();
    }
}
```

And a minimal TCP client:

```
include TCPSocket.con

class Main {
    Main() {
        try {
            var t=new TCPSocket();
            if (!t.Connect("localhost", 2000)) {
                echo "Error connecting to localhost";
                return -1;
            }
            // send data to the server
            t.Write("Hello Server!");
            // wait data from the server
            echo t.Read();
            t.Close();
        } catch (var exc) {
            echo exc;
            return -1;
        }
        return 0;
    }
}
```

The server should be run first, to wait for a new connection. Then, in a separate shell, run the client.

The server will show “Hello Server!” after the client connects and sends the data, and the client will print “Hello Client!” (received from the server).

In practice, the server will create a thread for every new connection, immediately after calling *Accept*.

When multiple network interfaces are available on a server, a socket may need to listen to a specific interface (see the *interface* parameter from the *Listen* member). A list containing all the available network interfaces can be obtained by calling:

```
array ListInterfaces();
```

The function will return an array of key-value arrays, having *ip, mask, mac, gateway, adapter, description, flags* and *type* as keys. On error it will return an empty array. This means that no network interfaces are available or some error occurred (you may want to check *errno()* for an error code).

A minimal interface query will look like:

```
import standard.net.socket

class Main {
    Main() {
        echo ListInterfaces();
    }
}
```

The output could look like:

```
Array {
    [0] =>
        Array {
            [0,"ip"] => 192.168.2.110
            [1,"mask"] => 255.255.255.0
            [2,"mac"] => 00:1C:D2:18:25:91
            [3,"gateway"] => 192.168.2.1
            [4,"adapter"] => {D882E90D-F8E8-429C-AAEB-A318943F26D7}
            [5,"description"] => Marvell Yukon 88E8056 PCI-E Gigabit
                Ethernet Controller
            [6,"flags"] => 1
            [7,"type"] => 6
        }
    [1] =>
        Array {
            [0,"ip"] => 169.254.180.206
            [1,"mask"] => 255.255.0.0
            [2,"mac"] => 00:50:56:C0:00:01
            [3,"gateway"] => 0.0.0.0
            [4,"adapter"] => {1E092817-81D3-4CA1-9FDC-AA5850AACA8C}
            [5,"description"] => VMware Virtual Ethernet Adapter for
                VMnet1
            [6,"flags"] => 0
            [7,"type"] => 6
        }
}
```

Note that flags and type are operating system-dependent and should be ignored. The *ip* value may be used with *Listen*, forcing the server to use the given interface.

For example, if the 6th line of the server socket example will be replaced with:

```
        if (t.Listen(2000, "192.168.2.110")) {
```

The server will listen for connections request coming only on the network interface having the 192.168.2.110 IP.

Note that IPs are not limited to IPv4. You could also listen on an IPv6-enabled interface.

11.2 UDP sockets

UDP, short for User Datagram Protocol enables two programs to send messages, called datagrams, over a network, without prior communications to set up special transmission channels or data paths. It has no guarantee that a datagrams will arrive orderly, or arrive at all.

UDP sockets are great for real-time applications, like VoIP or IpTV, where error checking is not necessary.

The *UDPSocket* class, defined in UDPSocket.con manages the datagrams send between sockets. The server socket must call *Bind* for the given port.

UDPSocket members:

UDPSocket (number is_ipv6=false)
Constructor, initializes an *UDPSocket*. For IPv6 sockets, the *is_ipv6* flag must be set to **true**.

Bind (number port, interface="")
Used in server sockets, bind the socket to the given port and interface. If interface is not set, all available interfaces will accept packets. Returns 0 if succeeded.

Close ()
Closes the socket

Read (var udphost=null, var udpport=null, max_size=0xFFFF)
Reads at most max_size bytes and returns the read buffer. On success, sets the *udphost* and the *udpport* to the sender host, respectively port. In case of error, it throws an error string.

Write (string buffer, string udphost, number port)
Sends buffer on the socket to the given *udphost* and *port*. Returns the number of bytes sent. In case of error, it throws an error string.

SetOption (number level, number option, number val)
Sets an option for socket. *level* may be SOL_SOCKET. *option* is a socket option (see TCP/IP options) and *val* the new value. Returns 0 on success, -1 on error.

GetOption (number level, number option, var val)
Gets an option for socket. *level* may be SOL_SOCKET. *option* is a socket option (see bellow). Returns 0 on success and sets the val to the value, -1 on error.

Ipv6 : boolean read-only property
Returns true if socket uses IPv6.

HasData : boolean property
Returns true if data can be read, or a connection event occurred.

Info : array property
Returns an array containing the IP and the port used by the current socket. If the socket is invalid, it will return an empty array.

Socket : number property
Returns the socket file descriptor for use with the low-level APIs

Error : number property
Returns the error code of the last socket operation

Note that there is no guarantee that a successfully call to *Write* will actually deliver the data to the remove host.

A minimal UDP server:

```
include UDPSocket.con

class Main {
    Main() {
        var t = new UDPSocket();
        if (t.Bind(2000)) {
            echo "Error in bind\n";
            return -1;
        }


        while (true) {
            try {
                // we need the host and port to send the message back
                var msg=t.Read(var host, var port);
                echo msg;
                t.Write("Hello Client!", host, port);
            } catch (var exc) {
                echo exc;
                break;
            }
        }
        t.Close();
    }
}
```

And a minimal UDP client:

```
include UDPSocket.con

class Main {
    Main() {
        try {
            var t=new UDPSocket();
            t.Write("Hello Server!", "localhost", 2000);
            echo t.Read();
            t.Close();
        } catch (var exc) {
            echo exc;
            return -1;
        }
        return 0;
    }
}
```

The server should be run first, to way for a new connection. Then, in a separate shell, run the client.

The server will show “Hello Server!” after the client connects and sends the data, and the client will print “Hello Client!” (received from the server).

11.3 UNIX sockets

A Unix domain socket or IPC socket (inter-process communication socket) is a data communications endpoint for exchanging data between processes executing within the same host operating system. This kind of socket is not available on Windows and is emulated using named pipes.

The *UNIXSocket* class, defined in UNIXSocket.con, has similar behavior with TCP connection, although you could create a UDP-like unix socket via low-level APIs. For establishing a connection, a client socket must connect to a unix domain socket listening and accepting connections.

UNIXSocket members:

Connect (string socketname)
Used in client sockets, connects to the unix socket named *socketname*. Returns true if succeeded, false if failed.

Listen (string socketname)
Used in server sockets, creates and listens on the given *socketname*. Returns 0 if succeeded.

Accept (return_static_socket=false)
Waits for a connection on a server socket. Returns an UNIXSocket object if return_static_socket is false, or a socket file descriptor as a number otherwise.

Close ()
Closes the socket

Read (max_size=0xFFFF)
Reads at most max_size bytes and returns the read buffer. In case of error, it throws an error string.

Write (string buffer)
Sends buffer on the socket. Returns the number of bytes sent. In case of error, it throws an error string.

SetOption (number level, number option, number val)
Sets an option for socket. *level* may be SOL_SOCKET. *option* is a socket option (see TCP/IP options) and *val* the new value. Returns 0 on success, -1 on error.

GetOption (number level, number option, var val)
Gets an option for socket. *level* may be SOL_SOCKET. *option* is a socket option (see bellow). Returns 0 on success and sets the val to the value, -1 on error.

HasData : boolean property
Returns true if data can be read, or a connection event occurred.

Socket : number property
Returns the socket file descriptor for use with the low-level APIs

Error : number property
Returns the error code of the last socket operation

Every client socket must first make a successfully call to *Connect* in order to perform *Read* or *Write* operations.

For the socket server you cannot call *Read* or *Write* directly. The function call order is: *Listen*, client_socket = *Accept*(), client_socket.*Read* and/or client_socket.*Write*. A call to Listen will fail, if another server program running on the same machine, listens on the same port.

Unix domain sockets use the file system as their address name space, for every listening socket, a special file will be created (in our example "mytestscoket" in the current directory).

A minimal UNIX domain socket server:

```
include UNIXSocket.con

class Main {
    Main() {
        var t = new UNIXSocket();
        if (t.Listen("./mytestsocket")) {
            echo "Error in listen\n";
            return -1;
        }


        while (true) {
            try {
                var client=t.Accept();
                if (!client)
```

```
                    break;

                echo client.Read();
                client.Write("Hello client!");
                client.Close();
            } catch (var exc) {
                echo exc;
                break;
            }
        }
        t.Close();
    }
}
```

And a minimal UNIX domain socket client:

```
include UNIXSocket.con

class Main {
    Main() {
        try {
            var t=new UNIXSocket();
            if (!t.Connect("./mytestsocket")) {
                echo "Error connecting to unix socket";
                return -1;
            }
            // send data to the server
            t.Write("Hello Server!");
            // wait data from the server
            echo t.Read();
            t.Close();
        } catch (var exc) {
            echo exc;
            return -1;
        }
        return 0;
    }
}
```

The above examples will work on Microsoft Windows, but internally will use named pipes.

The server should be run first, to wait for a new connection. Then, in a separate shell, run the client.

The server will show “Hello Server!” after the client connects and sends the data, and the client will print “Hello Client!” (received from the server).

In practice, the server will create a thread for every new connection, immediately after calling *Accept*.

Note that there is no guarantee that a call to *Write* will send the entire data buffer. It is important to analyze the result of the Write (the amount of bytes written), and then call Write again for the remaining buffer. This may happen also with *Read*, that may return less that the *max_size*, and a second call to *Read* may be needed. Due to socket send/receive buffer size, a big buffer written on the socket, may be actually split in multiple network packets.

11.4 Multicast sockets

In computer networking, multicast (one-to-many or many-to-many distribution) is group communication where information is addressed to a group of destination computers simultaneously. IP multicast is a method of sending Internet Protocol (IP) datagrams to a group of interested receivers in a single transmission. It is often employed for streaming media applications on the Internet and private networks. The method is the IP-specific version of the general concept of multicast networking. It uses specially reserved multicast address blocks in IPv4 and IPv6[2].

The *MulticastSocket* class, defined in MulticastSocket.con handles the datagrams received from a given source between sockets.

MulticastSocket members:

MulticastSocket (string host, number port)
Constructs the UDP multicast client socket using *host:port* as source.

Join (string host, interface=“””)
Joins a multicast group[3]. Returns true if succeeded or false if failed.

[2] http://en.wikipedia.org/wiki/IP_multicast on August 1st, 2014

[3] Source-specific multicast, see http://en.wikipedia.org/wiki/Source-specific_multicast

Drop (string host, interface="")

Drops a multicast group. Returns true if succeeded or false if failed

Close ()

Closes the socket

Read (var udphost=null, var udpport=null, max_size=0xFFFF)

Reads at most max_size bytes and returns the read buffer. On success, sets the *udphost* and the *udpport* to the sender host, respectively port. In case of error, it throws an error string.

Write (string buffer, string udphost, number port)

Sends buffer on the socket to the given *udphost* and *port*. Returns the number of bytes sent. In case of error, it throws an error string.

SetOption (number level, number option, number val)

Sets an option for socket. *level* may be SOL_SOCKET. *option* is a socket option (see TCP/IP options) and *val* the new value. Returns 0 on success, -1 on error.

GetOption (number level, number option, var val)

Gets an option for socket. *level* may be SOL_SOCKET. *option* is a socket option (see bellow). Returns 0 on success and sets the val to the value, -1 on error.

HasData : boolean property

Returns true if data can be read, or a connection event occurred.

Info : array property

Returns an array containing the IP and the port used by the current socket. If the socket is invalid, it will return an empty array.

Socket : number property

Returns the socket file descriptor for use with the low-level APIs

Error : number property

Returns the error code of the last socket operation

Bellow is a complete example that uses unix and multicast sockets.
Note that this program must be executed using the multi-threaded Concept core.

```
include CircularBuffer.con
include HTTPServer.con
include MulticastSocket.con
include UNIXSocket.con

class MulticastProxy {
    var source;
    var port;
    var timeout;
    var SocketName="";


    MulticastProxy(source, port, cname="default", timeout=30) {
        this.source=source;
        this.port=port;
        this.timeout=timeout;
        this.SocketName="/tmp/${cname}.sock";
    }

    Run() {
        var unix=new UNIXSocket();
        var sockname=this.SocketName;

        _unlink(sockname);
        unix.Listen(sockname);

        var sock=unix.Accept();
        if (!sock) {
            echo "Error in UNIX socket/Accept\n";
            return;
        }

        var m=new MulticastSocket(source, port);
        var last = time();
        var timeout = this.timeout;
        var source=this.source;
        // join the multicast group
        if (m.Join(source)) {
            while (true) {
                var data = m.Read();
                sock.Write(data);
                var now=time();
                if (now-last>=timeout) {
                    // rejoin every timeout seconds
                    if (!m.Drop(source))
                        echo "Error dropping\n";
                    else
                    if (!m.Join(source))
                        echo "Error in rejoin\n";
                    else
                        echo "Rejoined\n";
                    last=now;
                }
            }
        } else
            echo "Error joining multicast source\n";
    }
}

class Main {
    var[] buffers;
    var Lock;

    SafeBuffers() {
        Lock.Lock();
        var len=length buffers;
        var[] final;
        for (var i=0;i<len;i++) {
            var buf=buffers[i];
            if (buf)
                final[length final]=buf;
        }
        Lock.Unlock();
        return final;
    }

    RemoveBuffer(buffer) {
        Lock.Lock();
        var len=length buffers;
        var[] final;
        for (var i=0;i<len;i++) {
            var buf=buffers[i];
            if ((buf) && (buf!=buffer))
                final[length final]=buf;
        }
        buffers=final;
        Lock.Unlock();
    }

    OnRequest(child, method, file, protocol, headers) {
        if (method=="GET") {
            var buffer;
            try {
                child.Write("$protocol 200 OK\r\nContent-type:
                    video/MP2T\r\n");
                child.Write("\r\n");

                Lock.Lock();
                buffer = new CircularBuffer();
                var idx = length buffers;
                buffers[idx]=buffer;
                Lock.Unlock();

                while (true) {
                    var data=buffer.Get();
                    if (data)
                        child.Write(data);
                    else
                        Sleep(100);
                }
            } catch (var exc) {
                if (buffer)
                    RemoveBuffer(buffer);
                echo "Disconnected\n";
            }
            return 200;
        }
    }

    Loop(source, program) {
        var file = new File("r");
        // extract ts from multiple ts using ffmpeg
        file.Name = "ffmpeg -i $source -map 0:p:$program -vcodec
            copy -acodec copy -c:s copy -f mpegts -";
        // open as pipe
        file.POpen();
        while (true) {
            if (file.Read(var data, 0xFFFF)<=0)
                break;

            // copy to local variable
            var buffers = SafeBuffers();
            var len = length buffers;
            for (var i=0;i<len;i++) {
                var buffer=buffers[i];
                if (buffer)
                    buffer.Add(data);
            }
        }
        file.Close();
    }

    Main() {
        try {
            var ip="227.10.11.2";
            var port=1234;
            var program=410;

            var channel="HBO HD";
            // use it as a semaphore to avoid
            // concurrent writes
            Lock = new ReentrantLock();

            var h = new HTTPServer();
            // handle the request ourselves
            h.OnRequest=this.OnRequest;
            h.Start(8000);

            // create a proxy (multicast to unix socket)
            var m=new MulticastProxy(ip, port, channel);
            RunThread(m.Run);

            Loop("unix://"+m.SocketName, program);
        } catch (var exc) {
            echo "Exception: $exc\n";
        }
    }
}
```

The previous example creates a HTTP sever that waits for connections on port 8000, then reads from a multicast source using a *MulticastSocket*, outputs the data to a *UNIXSocket*, extracts a specific program from a MPTS (multiple program transport stream), coverts it to a SPTS (single program transport stream) using an open source video processor (*ffmpeg*) and then streams it to multiple HTTP clients. The *UNIXSocket* is used as an IPC method for communicating with *ffmpeg*.

Multicast sockets are available on Concept Application Server 3.0 or higher.

11.5 SSL/TLS communications

Transport Layer Security (TLS) and Secure Sockets Layer (SSL), are cryptographic protocols which are designed to provide communication security over insecure networks. They use X.509 certificates and hence asymmetric cryptography to assure the counterparty with whom they are communicating, and to exchange a symmetric key.

The TLS/SSL class is *TLSSocket*, a subclass of *TCPSocket* (inheriting all its methods) and define in TLSSocket.con. Note that secure sockets can be used with any type of socket, not just TCP. For UDP and UNIX domain sockets, SSL/TLS is available only through low-level APIs, defined in *standard.net.tls*.

TLSSocket specific members (see TCPSocket for inherited members):

TLS : number property
Returns the TLS handle for using it with low level APIs

CertificateInfo : array property
Returns a key-value array describing a certificate, with the following keys: "subject" (array), "issuer" (array), serialNumber (hex string, uppercase), "issued" (date, YYMMDDHHMMSSZ or YYYYMMDDHHMMSSZ), "expires" (date), "sha1" (hex string, uppercase), "md5" (hex string, uppercase), "publicKey" (binary string).

TLSError : number property
Returns the last TLS error code (or 0 if no error)

TLSErrorString : string property
Returns a human readable string describing the last error

TrustFile : string property
Sets the verify location CA file

TrustPath : string property

Sets the verify location CA directory

LoadKeys (string certfilename, string keyfilename)
Loads certificate and key from the given files. Returns 0 on succes, -1 if file named *certfilename* is invalid, -2 if file named *keyfilename* is invalid or -3 if key pair verification failed.

LoadKeysBuffer (string cert, string key)
Loads certificate and key from the given string buffers. Returns 0 on succes, -1 if *cert* is invalid, -2 if *key* is invalid or -3 if key pair verification failed.

Verify ()
Verifies the certificate returns a x509 certificate verification status.

AddCA (string certificate)
Adds a client CA(Certificate Authority) from the *certificate* buffer.

The *TLSSocket* class was designed to be a drop-in replacement for *TCPSocket*.

Every client socket must first make a successfully call to *Connect* in order to perform *Read* or *Write* operations.

For the socket server you cannot call *Read* or *Write* directly. The function call order is: *Listen*, client_socket = *Accept*(), client_socket.*Read* and/or client_socket.*Write*. A call to Listen will fail, if another server program running on the same machine, listens on the same port(TCP).

A minimal TLS web server:

```
include TLSSocket.con

class Main {
    Main() {
        var t=new TLSSocket();
        if ((!t.Listen(443)) && (!t.LoadKeys("publickey.cer",
            "privatekey.pem"))) {
            while (true) {
                try {
                    var child = t.Accept();
                    while (child.HasData>0)
                        echo child.Read();
                    child.Write("HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\n\r\nHello world!");
                    child.Close();
                } catch (var exc) {
                    echo exc;
                }
            }
        } else
            echo t.TLSError;
    }
}
```

Note the keys. The keys were generated used openssl:

```
openssl genrsa -out privatekey.pem 2048
```

```
openssl req -new -x509 -key privatekey.pem -out publickey.cer -days 1825
```

Optionally:

```
openssl pkcs12 -export -out public_privatekey.pfx -inkey privatekey.pem -in publickey.cer
```

A minimal TLS client:

```
include TLSSocket.con

class Main {
    Main() {
        var t=new TLSSocket();
        try {
            t.TrustPath="ssl";
            if (t.Connect("localhost", 443)) {
                switch (t.Verify()) {
                    case X509_V_OK:
                        break;
                    case X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT:
                    case X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN:
                        echo "Self-signed certificate\n";
                        break;
                    case X509_V_ERR_CERT_HAS_EXPIRED:
```

```
                    case X509_V_ERR_CRL_HAS_EXPIRED:
                    case X509_V_ERR_CERT_NOT_YET_VALID:
                    case X509_V_ERR_CRL_NOT_YET_VALID:
                        echo "Certificate expired or not yet valid\n";
                        break;
                    default:
                        break;
                }
                t.Write("GET /\r\n");
                echo t.Read();
                t.Close();
            }
        } catch (var exc) {
            echo exc;
        }
    }
}
```

The output:

```
Self-signed certificate
HTTP/1.1 200 OK
Content-Type: text/plain

Hello world!
```

The server should be run first, to wait for a new connection. Then, in a separate shell, run the client. Instead of running the client, a web browser could be used to navigate to https://localhost. Note that the browser will generate an alert for self-signed certificate.

In practice, the server will create a thread for every new connection, immediately after calling *Accept*. All rules and member order-of-call are identical with TCP/IP sockets.

By default, the server and client will negotiate the safest protocol between SSLv2, SSLv3 and TLS. The Concept Framework secure socket APIs are based on OpenSSL (libssl).

11.6 Bluetooth sockets

Bluetooth is a wireless technology standard for exchanging data over short distances (using short-wavelength radio waves in the ISM band from 2.4 to 2.485 GHz) from fixed and mobile devices, and building personal area networks (PANs).[4]

Bluetooth sockets are similar to the TCP sockets. On the server side, a socket will call Listen and wait for a connection, while on the client side, Connect will be called. Instead of IP addresses, the hardware address of the device will be used. The hardware address is identical with network MAC used by the IP sockets. A media access control address (MAC address) is a unique identifier assigned to network interfaces for communications on the physical network segment.[5].

Concept Framework supports 3 bluetooth protocols:

L2CAP

L2CAP is used within the Bluetooth protocol stack. It passes packets to either the Host Controller Interface (HCI) or on a hostless system, directly to the Link Manager/ACL link. L2CAP is used to communicate over the host ACL link. Its connection is established after the ACL link has been set up.

In basic mode, L2CAP provides packets with a payload configurable up to 64 kB, with 672 bytes as the default MTU, and 48 bytes as the minimum mandatory supported MTU. In retransmission and flow control modes, L2CAP can be configured for reliable or asynchronous data per channel by performing retransmissions and CRC checks. Reliability in either of these modes is optionally and/or additionally guaranteed by the lower layer Bluetooth BDR/EDR air interface by configuring the number of retransmissions and flush timeout (time after which the radio will flush packets). In-order sequencing is guaranteed by the lower layer.

RFCOMM

The Bluetooth protocol RFCOMM is a simple set of transport protocols, made on top of the L2CAP protocol, providing emulated

[4] http://en.wikipedia.org/wiki/Bluetooth on February 25, 2014

[5] http://en.wikipedia.org/wiki/MAC_address

RS-232 serial ports (up to sixty simultaneous connections to a Bluetooth device at a time). RFCOMM is sometimes called serial port emulation. RFCOMM provides a simple reliable data stream to the user, similar to TCP. Many Bluetooth applications use RFCOMM because of its widespread support and publicly available API on most operating systems. Additionally, applications that used a serial port to communicate can be quickly ported to use RFCOMM. In the protocol stack, RFCOMM is bound to L2CAP[6].

Service discovery protocol (SDP)
Used to allow devices to discover what services each other support, and what parameters to use to connect to them. For example, when connecting a mobile phone to a Bluetooth headset, SDP will be used to determine which Bluetooth profiles are supported by the headset (headset profile, hands free profile, advanced audio distribution profile, etc.) and the protocol multiplexer settings needed to connect to each of them. Each service is identified by a Universally Unique Identifier (UUID)

Concept Framework implements the *BTSocket* class, defined in *BTSocket.con* supporting both the bluetooth sockets and basic cross-platform SDP functions.

BTSocket members:

BTSocket (type = BTPROTO_RFCOMM)
Constructor. *type* is the protocol type to use with the socket. Valid values are BTPROTO_RFCOMM and BTPROTO_L2CAP.

Connect (string hostandchannel, string service = "00000000-0000-0000-0000-000000000000")
Used in client sockets, connects the server at given *hostandchannel* in the form *(host):channel*, for example (AA:BB:CC:DD:EE:FF):1. Returns true if succeeded, false if failed.

Listen (number channel, number maxconnections = 0xFF, interface="""")
Used in server sockets, listens on the given channel and interface. If interface is not set, will listen on all available interfaces. *maxconnections* specifies the maximum concurrent connections. Returns 0 if succeeded.

[6] http://en.wikipedia.org/wiki/Bluetooth_protocols on Febrary 25, 2014

Accept (return_static_socket=false)

Waits for a connection on a server socket. Returns a BTSocket object if return_static_socket is false, or a socket file descriptor as a number otherwise.

Close ()

Closes the socket

Read (max_size=0xFFFF)

Reads at most max_size bytes and returns the read buffer. In case of error, it throws an error string.

Write (string buffer)

Sends buffer on the socket. Returns the number of bytes sent. In case of error, it throws an error string.

SetOption (number level, number option, number val)

Sets an option for socket. Returns 0 on success.

GetOption (number level, number option, var val)

Gets an option for socket. Returns 0 on success and sets the val to the value, -1 on error.

HasData : boolean property

Returns true if data can be read, or a connection event occurred.

Info : array property

Returns an key-value array containing the hardware address ("address" key) and the channel ("port" key) used by the current socket. If the socket is invalid, it will return an empty array.

Socket : number property

Returns the socket file descriptor for use with the low-level APIs

Error : number property

Returns the error code of the last socket operation

static boolean Register (string servicename, string service_uuid, number port=-1, comments="")

Register(SDP) a service called *servicename* on the given *port*/channel. See bellow for a complete list with all the services. Returs true on success, false if failed.

static boolean Unregister (string servicename, string service_uuid, number port=-1, comments="")

Unregisters(SDP) a previously registered service named *servicename* (all other parameters are ignored). Returs true on success, false if failed.

static array Discover (string service_type_uuid="")

Scan for a list of all devices and all the available services. If *service_type_uuid* is set, it will only scan for services matching the given UUID (universally unique identifier). An UUID is held as a string in the {*00000000-0000-0000-0000-000000000000*} format. If no bluetooth adapter is available on the host machine, it will return *null*. If no devices are in range, it will return an empty array. If devices are in range, it will return an array, having the following structure:

```
Array {
    [0] =>
        Array {
            [0,"Name"] => Device Name
            [1,"Comment"] =>
            [2,"Address"] => (5C:E8:EB:40:C5:6D)
            [3,"Services"] =>
                Array {
                    [0] =>
                        Array {
                            [0,"Name"] => OBEX File Transfer
                            [1,"Comment"] =>
                            [2,"Service"] =>
                                {00000000-0000-0000-0000-000000000000}
                            [3,"Address"] => (5C:E8:EB:40:C5:6D):5
                        }
                    [1] =>
                        Array {
                            [0,"Name"] => Advanced Audio
                            [1,"Comment"] =>
                            [2,"Service"] =>
                                {00000000-0000-0000-0000-000000000000}
                            [3,"Address"] =>
                                (5C:E8:EB:40:C5:6D):25
                        }
                    ...
                }
        }
    [1] =>
```

```
       Array {
           [0,"Name"] => Device Name 2
           [1,"Comment"] =>
           [2,"Address"] => (E4:32:CB:C7:A9:F3)
       }
    ...
}
```

Predefined values to be used as service UUID are:

ServiceDiscoveryServer_UUID, BrowseGroupDescriptor_UUID SerialPort_UUID, LANAccessUsingPPP_UUID, DialupNetworking_UUID, IrMCSync_UUID, OBEXObjectPush_UUID, OBEXFileTransfer_UUID, IrMCSyncCommand_UUID, Headset_UUID, CordlessTelephony_UUID, AudioSource_UUID, AudioSink_UUID, AVRemoteControlTarget_UUID, AdvancedAudioDistribution_UUID, AVRemoteControl_UUID, VideoConferencing_UUID, Intercom_UUID, Fax_UUID, HeadsetAudioGateway_UUID, WAP_UUID, WAPClient_UUID, PANU_UUID, NAP_UUID, GN_UUID, DirectPrinting_UUID, ReferencePrinting_UUID, Imaging_UUID, ImagingResponder_UUID, ImagingAutomaticArchive_UUID, ImagingReferenceObjects_UUID, Handsfree_UUID, HandsfreeAudioGateway_UUID, DirectPrintingReferenceObjects_UUID, ReflectedUI_UUID, BasicPringing_UUID, PrintingStatus_UUID, HumanInterfaceDevice_UUID, HardcopyCableReplacement_UUID, HCRPrint_UUID, HCRScan_UUID, CommonISDNAccess_UUID, VideoConferencingGW_UUID, UDIMT_UUID, UDITA_UUID, AudioVideo_UUID, SIMAccess_UUID, PnPInformation_UUID, GenericNetworking_UUID, GenericFileTransfer_UUID, GenericAudio_UUID and GenericTelephony_UUID

Every client socket must first make a successfully call to *Connect* in order to perform *Read* or *Write* operations.

For the socket server you cannot call *Read* or *Write* directly. The function call order is: *Listen*, client_socket = *Accept*(), client_socket.*Read* and/or client_socket.*Write*. A call to Listen will fail, if another server program running on the same machine, listens on the same channel. The function call sequence is exactly the same as for *TCPSocket*.

A minimal Bluetooth server:

```
include BTSocket.con

```

```
class Main {
    Main() {
        var srv_name = "My test service";
        var s=new BTSocket();
        // listen on channel 1, accept max one connection
        if (s.Listen(1, 1)) {
            echo "Error initializing server";
            return;
        }
        echo "Bluetooth server initialized\n";

        // make the service discoverable (as serial port service)
        if (BTSocket.Register(srv_name, SerialPortService_UUID))
            echo "Service $srv_name successfully registered\n";

        while (true) {
            var client = s.Accept();
            echo client.Read();
            client.Write("Hello !");
        }
        s.Close();
        BTSocket.Unregister(srv_name);
    }
}
```

And a minimal Bluetooth client, that will scan for devices and then send “Hello !” to all devices having a service named “My test service”.

```
include BTSocket.con

class Main {
    Main() {
        try {
            var s=new BTSocket();
            var devices = s.Discover();
            if (devices) {
                var len = length devices;
                for (var i=0;i<len;i++) {
                    var device=devices[i];

                    var services=device["Services"];
                    if (services) {
                        var device_name=device["Name"];
                        var len2=length services;
                        for (var j=0;j<len2;j++) {
                            var service=services[j];
                            if ((service) && (service["Name"]=="My
                                test service")) {
                                var addr_and_port=service["Address"];
                                if (addr_and_port) {
                                    echo "Connecting to $device_name
                                        [$addr_and_port]\n";
                                    if (s.Connect(addr_and_port)) {
                                        echo "Connected !";
                                        s.Write("Hello !");
                                        echo s.Read();
                                    }
                                }
                            }
                        }
                    }
                }
            }
        } catch (var exc) {
            echo exc;
            return -1;
        }
        return 0;
    }
}
```

The server should be run first, to wait for a new connection. Then, on a separate machine, run the client.

11.7 Cryptographic functions

Concept Framework supports a set of frequent used cryptographic and hashing algorithms, by importing *standard.lib.cripto*. Note that cripto is misspelled. By the time it was noticed, there was already a consistent set of applications using this library, so I decided not to change it, because Concept is and will be backwards-compatible, meaning that an application written for Concept 1.0 will run with no problem on CAS 3.0.

The cryptographic and hashing static functions are optimized for high

CPU efficiency. It includes algorithms like RSA, AES, MD5, SHA1, SHA256, Murmur or PBKDF2.

High level *standard.lib.cripto* functions:

md5 (string buffer)
Computes the md5 hash for the *buffer* and returns the value as a hexadecimal string (lowercase)

sha1 (string buffer)
Computes the sha1 hash for the *buffer* and returns the value as a hexadecimal string (lowercase)

sha256 (string buffer)
Computes the sha256 hash for the *buffer* and returns the value as a hexadecimal string (lowercase)

crc32 (string buffer)
Computes the crc32 for the *buffer* and returns the value as a number (32 bit)

Murmur (string buffer)
Computes the Murmur hash for the *buffer* and returns the value as a number (32 bit)

PBKDF2 (string password, string salt, key_size=16, rounds=1000)
Returns the generated derived key as a string buffer. *password* is the master password from which a derived key is generated, *salt* is a cryptographic salt. *key_size* is the desired length of the derived key. *rounds* is the number of iterations desired. *DRMKey*()
Returns the private symmetric key used in concept protocol negotiation. This is useful for sending encrypted VoIP packages over unsecured connections.

hmac_md5 (string key, string message, var outhash)
Computes the MAC(message authentication code) using MD5 as hash function, and outputs in *outhash* as a lowercase hex string. Returns 0 on success.

hmac_sha1 (string key, string message, var outhash)
Computes the MAC using SHA1 as hash function, and outputs in *outhash* as a lowercase hex string. Returns 0 on success.

hmac_sha256 (string key, string message, var outhash)
Computes the MAC using SHA256 as hash function, and outputs in *outhash* as a lowercase hex string. Returns 0 on success.

Low level hashing functions:

MD5Init ()
Returns an md5 context handle

MD5Update (context, string data)
Appends (and updates)data to the md5 context

MD5Final (context)
Frees the md5 context and returns the hash a lowercase hex string

SHA1Init ()
Returns an sha1 context handle

SHA1Update (context, string data)
Appends (and updates)data to the sha1 context

SHA1Final (context)
Frees the sha1 context and returns the hash a lowercase hex string

SHA256Init ()
Returns an sha256 context handle

SHA256Update (context, string data)
Appends (and updates)data to the sha256 context

SHA256Final (context)
Frees the sha256 context and returns the hash a lowercase hex string

AES (Advanced Encryption Standard) is an symmetrical encryption algorithm. This means that the same key is used both from encryption and decryption. Concept uses a very fast implementation of the AES algorithm, allowing programs to encrypt/decrypt data in real-time, without wasting significant CPU power.

AES Cryptographic functions:

AESEncryptInit (string key)
Key must be 128, 192 or 256 bits in length (16 bytes, 24 bytesm respectively 32 bytes). Returns an AES context handle

AESEncrypt (context, string data, mode=BLOCKMODE_ECB, force_PCKS7_padding=false)
Mode may by BLOCKMODE_ECB or BLOCKMODE_CBC. If *force_PCKS7_padding* is set to true, complete packets will be padded.

AESDecryptInit (string key)
Key must be 128, 192 or 256 bits in length (16 bytes, 24 bytesm respectively 32 bytes). Returns an AES context handle

AESDecrypt (context, string data, mode=BLOCKMODE_ECB, force_PCKS7_padding=false)
Mode may by BLOCKMODE_ECB or BLOCKMODE_CBC. If *force_PCKS7_padding* is set to true, complete packets will be padded.

AESRelease (context)
Frees the memory associated with an AES context

For other AES block modes (128 bit only) see Concept Framework documentation for: *context AesSetKey(key), AesDone(context) , AesSetInitVector(context, string vector), AesSetFeedbackSize(context, size), string AesEncryptECB(context, buffer, force_PCKS7=false), AesEncryptCBC, AesEncryptPCBC, AesEncryptCRT, AesEncryptOFB, AesEncryptCFB, AesDecryptECB, string AesDecryptCBC(context, buffer, force_PCKS7=false), AesDecryptPCBC, AesDecryptCRT, AesDecryptOFB, AesDecryptCFB.*

RSA is an asymmetric encryption algorithm. This means that it uses two different keys, one for encrypting (called public key) and another for decrypting (the private key). It is not possible to use the public key for decrypting.

Basic RSA Cryptographic functions:

rsa_generate_keys (number keybits, var private_key, var public_key)
Generates a random *keybits* bits key pair and sets private_key and public_key. Returns true on success.

rsa_encrypt (string plain_text, string public_key)
Encrypts text with the given public key. Returns the encrypted string (cipher text).

rsa_decrypt (string cipher_text, string private_key)
Decrypts text with the given private key. Returns the plain text string.

rsa_sign (string plain_text, string private_key)
Signs text using the given private key. Returns the encrypted string (cipher text).

rsa_verify ()
Verifies text using the given public key. Returns plain text.

RSA and AES work great together. You can use RSA's public key/private key feature to safely exchange a symmetric AES key. Also, RSA uses significantly more resources than AES, and is limited to relatively small messages (up to 245 bytes for 2048 bit keys). The maximum RSA message size (in bytes) is KeySizeInBits / 8 - 11. On the other hand, AES has no message limit.

AES can use different modes. ECB (Electronic Codebook) and CBC (Cipher-Block Chaining) are supported for all key sizes. When in ECB mode, each plain text block in encrypted independently. In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. Figure 11.1 shows how an image(left) will appear after being ecrypted using ECB(center) and CBC(right). It doesn't mean that CBC is safer than ECB. It only means that ECB produces an output that is indistinguishable from random noise. When using CBC, if a data block is lost, the following blocks will not be decrypted correctly.

cryptoexample.con

```
import standard.lib.cripto

class Main {
    Encrypt(plaintext, password) {
        // 16 bytes = 128 bits key, 10000 rounds
        var key=PBKDF2(password, "SALT.Devronium", 16, 10000);

        var ctx=AESEncryptInit(key);
```

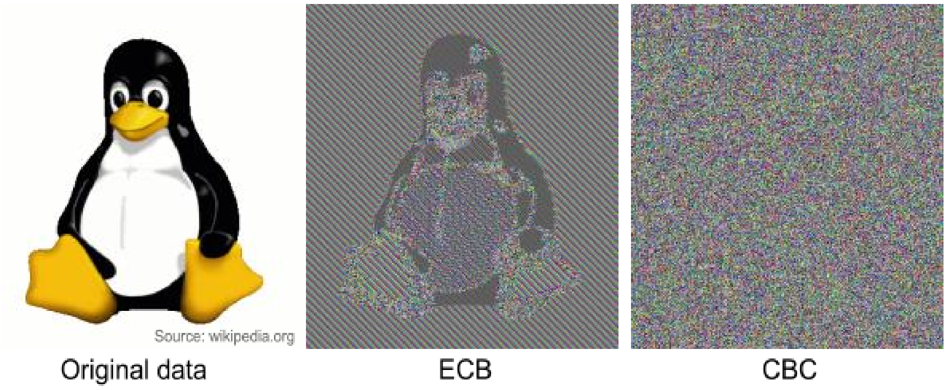


Figure 11.1:
AES modes

```
        var ciphertext=AESEncrypt(ctx, plaintext, BLOCKMODE_CBC,
            true);
        AESRelease(ctx);
        return ciphertext;
    }

    Decrypt(ciphertext, password) {
        // 16 bytes = 128 bits key, 10000 rounds
        var key=PBKDF2(password, "SALT.Devronium", 16, 10000);
        var ctx=AESDecryptInit(key);
        var plaintext=AESDecrypt(ctx, ciphertext, BLOCKMODE_CBC,
            true);
        AESRelease(ctx);
        return plaintext;
    }

    Main() {
        var cipher = this.Encrypt("Hello world !", "Some password");
        echo this.Decrypt(cipher, "Some password");
    }
}
```

cryptoexample.con will encrypt ”Hello world !” using AES-128 (see *Encrypt*) and then will decrypt the resulted cipher text using *Decrypt*. For using AES-192 or AES-256, the line PBKDF2(password, “SALT.Devronium”, 16, 10000) must be chganged to PBKDF2(password, “SALT.Devronium”, 24, 10000) or PBKDF2(password, “SALT.Devronium”, 32, 10000) in both functions.

rsaexample.con

```
import standard.lib.cripto

class Main {
   Main() {
       // generate a random key pair
       // may take up to a few seconds
       rsa_generate_keys(512, var priv, var pub);
       // encrypt
       var cipher = rsa_encrypt("Hello world !", pub);
       // decrypt
       echo rsa_decrypt(cipher, priv);
   }
}
```

The above example will generate a key pair, will encrypt ”Hello world !” using the public key and the decrypt it using the private key. Key generation is a relatively slow process, and generating random key pairs for every session will not be practical. In practice, the RSA keys will be pre-generated and stored on a disk.

Concept Framework provides a simple RSA class, defined in RSA.con. It is mostly a wrapper for the OpenSSL command line utility.

The previous example can be rewritten to use OpenSSL’s RSA implementation:

```
include RSA.con

class Main {
   Main() {
       try {
           if (!RSA.GenerateKeys(var pub, var priv, 512))
               return -1;

           var cipher = RSA.Encrypt(pub, "Hello world !");
           echo RSA.Decrypt(priv, cipher);
       } catch (var exc) {
           echo exc;
```

```
        }
    }
}
```

Note that you cannot decrypt using rsa_decrypt the output of RSA.Encrypt, or share keys between the two.

hashexample.con

```
import standard.lib.cripto

class Main {
    Main() {
        var text = "Hello World!";
        echo "MD5($text)\t:\n "+md5(text)+"\n";
        echo "SHA1($text)\t:\n "+sha1(text)+"\n";
        echo "SHA256($text)\t:\n "+sha256(text)+"\n";
        echo "CRC32($text)\t:\n "+crc32(text)+"\n";
        echo "Murmur($text)\t:\n "+Murmur(text)+"\n";
    }
}
```

The output:

```
MD5(Hello World!)     :
 ed076287532e86365e841e92bfc50d8c
SHA1(Hello World!)    :
 2ef7bde608ce5404e97d5f042f95f89f1c232871
SHA256(Hello World!)  :
 7f83b1657ff1fc53b92dc18148a1d65dfc2d4b1fa3d677284addd200126d9069
CRC32(Hello World!)   :
 472456355
Murmur(Hello World!)  :
 1624100388
```

Murmur hash is a very interesting non-cryptographic hash function due to its very fast execution time, being perfect for hash-based lookup. The Concept Core use this function internally, for faster lookups.

11.8 High level protocols

A set of high level protocols are available in Concept Framework.

Protocol	**Class**	**include/import**
SMTP	MSMTP	MSMTP.con
FTP	FtpConnection, FtpFile	Ftp.con
POP3	(static interface)	standard.net.mail
SIP	OPALSIP	OPALSIP.con
DNS	(static interface)	standard.net.dns
GeoIP	GeoIP	GeoIP.con
SNMP	SNMP	SNMP.con
IM	(static library)	standard.net.im
Modbus	(static library)	standard.net.modbus
RouterOS	(static library)	standard.net.routeros
SOAP	(static library)	standard.net.soap
Twitter	Twitter	Twitter.con
HTTP(s)	URL	URL.con
METAR	METAR	METAR.con
WolframAlpha	WolframAlpha	WolframAlpha.con
GoogleSearch	GoogleSearch	GoogleSearch.con

We will discuss some of them in detail, for the others, see the Concept Documentation.

For exmaple, Wolfram Alpha is a computational knowledge engine or answer engine developed by Wolfram Research. It is an online service that answers factual queries directly by computing the answer from externally sourced “curated data”, rather than providing a list of documents or web pages that might contain the answer as a search engine might.

simpleWolfram.con

```
include WolframAlpha.con
import standard.C.io

define WOLFRAM_KEY "replace_with_your_api_key"

class Main {
   function Main() {
      var w=new WolframAlpha(WOLFRAM_KEY);
```

```
        w.FetchRemoteContent=true;
        try {
            var result=w.Query("y^2=x^2 - x^4");
            var arr=result.Pods;
            var len=length arr;

            for (var i=0;i<len;i++) {
                var pod=arr[i];
                echo "$i:"+pod.Text;
                //if (pod.Sound)
                //  WriteFile(pod.SoundContent, "out$i.mid");
                //echo "\n";
            }
        } catch (var exc) {
            echo exc;
        }
    }
}
```

We asked to plot $y^2 = x^2 - x^4$, but in reality we can ask anything, for example "weather in Nepal". A full UI example can be found in the Samples directory of your Concept Distribution (WolframTest).

A protocol that gets used a lot by spammers si SMTP, short for Simple Mail Transfer Protocol. This protocol gets used every time you send an e-mail.

```
include MSMTP.con

class Main {
    Main() {
        var m=new MSMTP();
        m.Host="mail.spammer.com";
        m.From="master@spammer.com";
        m.To="victim@domain.com";
        //m.User="username";
        //m.Password="password";

        if (m.Send("Hello world !!!"))
            echo "Success!";
    }
}
```

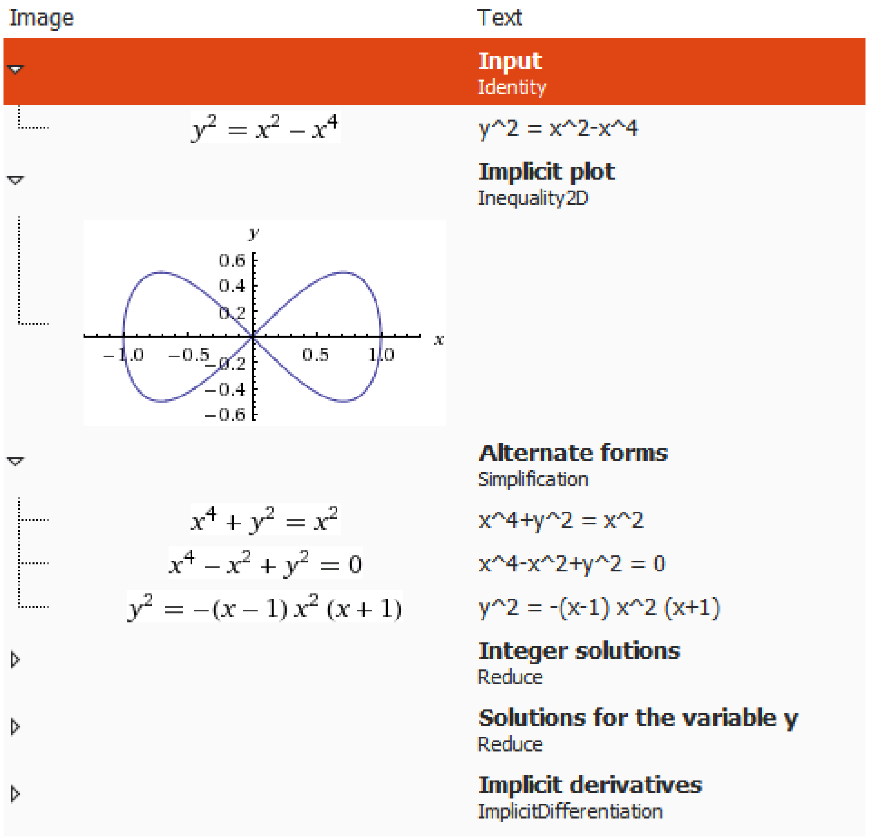


Image	Text
	Input Identity
$y^2 = x^2 - x^4$	y^2 = x^2-x^4
	Implicit plot Inequality2D
	Alternate forms Simplification
$x^4 + y^2 = x^2$	x^4+y^2 = x^2
$x^4 - x^2 + y^2 = 0$	x^4-x^2+y^2 = 0
$y^2 = -(x-1)\, x^2\, (x+1)$	y^2 = -(x-1) x^2 (x+1)
	Integer solutions Reduce
	Solutions for the variable y Reduce
	Implicit derivatives ImplicitDifferentiation

Figure 11.2:
Wolfram Alpha APIs

For a complete list of MSMTP's members, see Concept Framework documentation, topic MSMTP.

standard.coding.base64 provides functions for base64 encoding. Base64 is a group of similar binary-to-text encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation. The term Base64 originates from a specific MIME content transfer encoding.[7].

standard.coding.base64 provides two static functions:

```
string mime_encode(string binarydata);
string mime_decode(string base64data);
```

mime_encode returns the base64 encoded data of the *binarydata*. The resulting data can be then decoded using *mime_decode*.

METAR is a format for reporting weather information. A METAR weather report is predominantly used by pilots in fulfillment of a part of a pre-flight weather briefing, and by meteorologists, who use aggregated METAR information to assist in weather forecasting. METARs typically come from airports or permanent weather observation stations. Reports are generated once an hour or half-hour, but if conditions change significantly, a report known as a special (SPECI) may be issued.

The METAR class is defined in METAR.con and uses public data from the noaa.gov ftp server.

simplemetar.con

```
include METAR.con

class Main {
   Main() {
      // KJFK = John F. Kennedy International Airport, New York
      // look for an ICAO list for all available stations
      var metar=METAR::Get("KJFK");
      if (metar) {
         var strdata="Temperature in New York is
             ${metar.Temperature}C, wind ${metar.WindKH}Km/h";
         if (metar.Overcast)
```

[7] Source: http://en.wikipedia.org/wiki/Base64 on January 25, 2014

```
                strdata+=", cloudy";
            var modifier="";
            if (metar.Intensity==MODIFIER_HEAVY)
                modifier="heavy ";
            else
            if (metar.Intensity==MODIFIER_LIGHT)
                modifier="light ";

            if ((metar.Snow) || (metar.TempoSnow))
                strdata+=", ${modifier}snow";
            if ((metar.Rain) || (metar.TempoRain))
                strdata+=", ${modifier}rain";
            if ((metar.Thunderstorm) || (metar.TempoThunderstorm))
                strdata+=", thunderstorm";
            if (metar.Fog)
                strdata+="fog";

            echo strdata;
        }
    }
}
```

Output:

```
Temperature in New York is 4C, wind 26Km/h
```

Using the METAR data you can easily create an weather application. You just need a list with all the ICAO codes (the location indicator assigned by the International Civil Aviation Organization). Then, using GeoIP or location services, you locate the user, and look for the nearest station, and get the METAR data for that station. Also, METAR

standard.net.im implements protocols for instant messaging protocols like Bonjour, Gadu-Gadu, IRC, Lotus Sametime, MySpaceIM, MXit, MSNP, Novell GroupWise, OSCAR (AIM/ICQ/MobileMe), SIP/SIMPLE, SILC, XMPP/Jingle (Google Talk, LJ Talk, Gizmo5, Facebook Chat, ...), YMSG and Zephyr. For more information check Concept Framework documentation and libpurple APIs. The BasicPurpleIO class, defined in BasicPurpleIO.con provides a simple (and limited) way for exchanging instant messages.

simpleim.con

```
include BasicPurpleIO.con

class Main {
    var P;

    OnMessage(conv, who, alias, message, flags, mtime) {
        echo "$who> $message\n";
        if (who) {
            // echo back the message
            P.SendIm(who, message);
        }
    }

    OnFileTransfer(xfer, filename, filesize) {
        echo "Accepting $filename ($filesize bytes)\n";
        BasicPurpleIO::Accept(xfer, filename);
    }

    OnFinishTransfer(xfer) {
        echo "Done $xfer !\n";
    }

    OnSignedOn(gc) {
        echo P.Buddies;
        P.SendIm("yourbuddysid", "Hello! I just signed in");
    }

    Main() {
        P=new BasicPurpleIO();
        echo P.Protocols;
        P.OnSignedOn=OnSignedOn;
        P.OnMessage=OnMessage;
        P.OnFileTransfer=OnFileTransfer;
        P.OnFinishTransfer=OnFinishTransfer;
        P.Login("prpl-yahoo", "youryahooid", "yahoopassword");
        P.Go();
    }
}
```

Replace *youryahooid*, *yahoopassword* with your YahooID credentials, and *yourpuddysid* with a buddy to which a message will be send upon log-in.

You can just change the *prpl-yahoo* with *prpl-msn* to log in to MSN messenger. For a list of all the supported protocols, you should read the *BasicPurpleIO.Protocols* read-only property.

The *standard.net.modbus* import library implements support for the Modbus protocol, using *libmodbus*[8]. Modbus is a serial communications protocol originally published by Modicon (now Schneider Electric) in 1979 for use with its programmable logic controllers (PLCs). Simple and robust, it has since become a de facto standard communication protocol, and it is now a commonly available means of connecting industrial electronic devices[9].

The APIs implemented by *standard.net.modbus* are almost identical with the C APIs implemented by *libmodbus* (see libmodbus documentation at http://libmodbus.org/documentation/ for more information).

A minimal Modbus client is shown bellow:

modbus.con

```
import standard.net.modbus

class Main {
    GetModbusData(ip, reg_start, len, port=502) {
        var mb = modbus_new_tcp(ip, port);
        modbus_connect(mb);
        modbus_read_registers(mb, reg_start, len, var tab_reg);
        modbus_close(mb);
        modbus_free(mb);
        return tab_reg;
    }

    Main() {
        // read starting from register 7
        // (6, here, because is zero-based)
        // 4 values
        echo GetModbusData("192.168.1.130", 6, 4);
    }
}
```

Note that the *standard.net.modbus* simply wraps the libmodbus, so you will

[8] http://libmodbus.org/

[9] http://en.wikipedia.org/wiki/Modbus on March 2nd, 2014

need to explicitly call *modbus_close* and *modbus_free*. This is implemented this way for maintaining compatibility with already written C code.

11.9 SSH protocol

Secure Shell (SSH) is a cryptographic network protocol for secure data communication, remote command-line login, remote command execution, and other secure network services between two networked computers that connects, via a secure channel over an insecure network, a server and a client (running SSH server and SSH client programs, respectively).[10]

The *standard.net.ssh* import library provides the following SSH static functions:

handle SSHConnect (number socket[,var errcode])
Negotiates a connection withe the server on the given TCP socket (must be connected). If *errcode* is given, it will contain the error code of the operation (if failed). Returns a session handle if succeeded or null if failed.

string SSHFingerprint (handle session)
Returns the SSH host key as s string for the given session.

number SSHAuth (handle session, string username, string password)
Authenticates to the remote host on the initialized *session*, using *username* and *password*. Returns 0 if succeeds, non-zero if failed.

number SSHAuthPublicKey (handle session, string username, string pubfile, string privfile, string passphrase)
Authentificates to the remote SSH server using a public key, read from a file. *pubfile* is the path name of the public key file. (e.g. /etc/ssh/hostkey.pub), *privfile* is the path name of the private key file. (e.g. /etc/ssh/hostkey) and *passphrase* is the passphrase to use when decoding privatekey. Returns 0 if succeeds, non-zero if failed.

handle SSHChannelOpen (handle session)
Opens a SSH channel in the given *session*. Returns a channel handle is succeeded, or null if it fails.

[10] http://en.wikipedia.org/wiki/Secure_Shell on March 6, 2014

number SSHLastError (handle session)
Returns the error code of the last operation for the given *session*.

string SSHLastErrorString (handle session)
Returns the error description as a human readable string, of the last operation for the given *session*.

number SSHShell (handle channel)
Request a shell on a channel. Returns 0 if succeeds, non-zero if failed.

number SSHProcess (handle channel, string request, string message)
Request a shell on a channel. *request* is the process type to startup. The SSH2 protocol currently defines shell, exec, and subsystem as standard process services. *message* is the request specific message data to include. Returns 0 if succeeds, non-zero if failed.

number SSHExec (handle channel, string command)
Executes the given *command* on the opened *channel*. Returns 0 if succeeds, non-zero if failed.

string SSHRead (handle channel, number max_size)
Reads data from the channel and returns it. On error it will return an empty string.

number SSHWrite (handle channel, string buffer)
Writes *buffer* on the opened *channel*. Returns the number of bytes written, or a negative value if it failed.

number SSHSendEOF (handle channel)
Sends EOF on the given channel. Returns 0 if succeeded, non-zero if failed.

SSHSetKeepAlive (handle session, boolean want_replay, number seconds)
Sets the keep-alive seconds for the given session. If want_reply is set, a replay will be requested to the SSH server.

number SSHSendKeepAlive (handle session, var seconds)
Sends a keep alive packet and sets *seconds* to the number of seconds until the next keep alive packet. Returns 0 if succeeded, non-zero if failed.

SSHSetBlocking (handle session, boolean blocking)
Makes SSHRead non-blocking (if blocking is set to true).

number SSHChannelClose (handle channel)

Closes a SSH channel. Returns 0 if succeeded, non-zero if failed. If it fails, the channel is not freed.

number SSHDisconnect (handle session)

Closes a SSH session. Returns 0 if succeeded, non-zero if failed.

The following example will connect to a remote host running a SSH server, and then execute a “ls” command.

SSHclient.con

```
import standard.net.ssh
include TCPSocket.con

class Main {
    Main() {
        var sock=new TCPSocket();
        if (sock.Connect("yourhost.com", 22)) {
            var session=SSHConnect(sock.Socket, var rc);
            if (session) {
                var key = SSHFingerprint(session);
                // here you should check the key
                if (!SSHAuth(session, "username", "password")) {
                    echo "Authentificated!\n";
                    var channel = SSHChannelOpen(session);
                    if (channel) {
                        echo "Channel created!\n";
                        if (!SSHShell(channel)) {
                            echo "Executed!\n";
                            // run a "ls" command
                            SSHWrite(channel, "ls\n");
                            echo SSHRead(channel, 0xFFFF, 0);
                        }
                        SSHChannelClose(channel);
                    }
                } else
                    echo SSHLastErrorString(session);
                SSHDisconnect(session);
            }
            sock.Close();
        }
    }
}
```

yourhost.com should be replaced with a valid hostname running a SSH server on port 22. *username* and *password* must be replaced with a valid user name and password.

11.10 DNS and GeoIP

Every time a user connects to a named host, a DNS (Domain Name System) server will be queried. It basically translates a host and service to an IP address.

The DNS functions are defined in *standard.net.dns* import library.

There are only 3 functions that handle the entire DNS query process:

array DNSQuery (string name, class=ns_c_any, type=ns_t_any)
: Looks for *name* host. class may be *ns_c_in, ns_c_2, ns_c_chaos, ns_c_hs, ns_c_none* or *ns_c_any*, and *type* ns_t_a , *ns_t_ns, ns_t_md, ns_t_mf, ns_t_cname, ns_t_soa, ns_t_mb, ns_t_mg, ns_t_mr, ns_t_null, ns_t_wks, ns_t_ptr, ns_t_hinfo, ns_t_minfo, ns_t_mx, ns_t_txt, ns_t_rp, ns_t_afsdb, ns_t_x25, ns_t_isdn, ns_t_rt, ns_t_nsap, ns_t_nsap_ptr, ns_t_sig, ns_t_key, ns_t_px, ns_t_gpos, ns_t_aaaa, ns_t_loc, ns_t_nxt, ns_t_eid, ns_t_nimloc, ns_t_srv, ns_t_atma, ns_t_naptr, ns_t_kx, ns_t_cert, ns_t_a6, ns_t_dname, ns_t_sink, ns_t_opt, ns_t_apl, ns_t_ds, ns_t_sshfp, ns_t_rrsig, ns_t_nsec, ns_t_dnskey, ns_t_tkey, ns_t_tsig, ns_t_ixfr, ns_t_axfr, ns_t_mailb, ns_t_maila, ns_t_any* or *ns_t_zxfr*.
It returns an array containing the requested records.

array DNSResolve(string host)
: Resolves the given host

string DNSReversed(string ip)
: Performs a reversed query (returns the host by its ip).

The functions are IPv6 ready, and relatively straight forward:

```
import standard.net.dns

class Main {
   Main() {
        echo DNSQuery("www.google.com");
        echo "\n===\n";
        echo DNSReversed("8.8.8.8");
        echo "\n===\n";
        echo DNSResolve("www.google.ro");
    }
}
```

Output:

```
Array {
    [0] =>
        Array {
            [0,"type"] => A
            [1,"host"] => www.google.com
            [2,"addr"] => 173.194.35.177
        }
    [1] =>
        Array {
            [0,"type"] => A
            [1,"host"] => www.google.com
            [2,"addr"] => 173.194.35.176
        }
[...]
}
===
google-public-dns-a.google.com
===
Array {
    [0,"www.google.ro"] => 173.194.44.56
}
```

GeoIP is a IP/location database. It allows you to query an ip for location, organization, connection speed and user type. The interface class is GeoIP, defined in GeoIP.con.

```
include GeoIP.con

class Main {
    function Main() {
        var g=new GeoIP();
        g.Open("GeoLiteCity.dat");
        echo g.GetRecord("8.8.8.8");
    }
}
```

Outputs:

```
Array {
        [0,"country_code"] => US
        [1,"country_code3"] => USA
        [2,"country_name"] => United States
        [3,"region"] => CA
        [4,"city"] => Mountain View
        [5,"latitude"] => 37.4192008972168
        [6,"longitude"] => -122.057403564453
        [7,"metro_code"] => 807
        [8,"dma_code"] => 807
        [9,"area_code"] => 650
        [10,"charset"] => 0
        [11,"continent_code"] => NA
}
```

Note, that you must first download the GeoIP database from maxmind.com.

11.11 SNMP and network management

Simple Network Management Protocol (SNMP) is an ”Internet-standard protocol for managing devices on IP networks”. Devices that typically support SNMP include routers, switches, servers, workstations, printers, modem racks and more. It is used mostly in network management systems to monitor network-attached devices for conditions that warrant administrative attention. SNMP is a component of the Internet Protocol Suite as defined by the Internet Engineering Task Force (IETF).

SNMP itself does not define which information (which variables) a managed system should offer. Rather, SNMP uses an extensible design, where the available information is defined by management information bases (MIBs). MIBs describe the structure of the management data of a device subsystem; they use a hierarchical namespace containing object identifiers (OID). Each OID identifies a variable that can be read or set via SNMP.

The SNMP class, defined in SNMP.con handles the protocol using the following members:

Handle : number property
Handle for low-level APIs

Error : string property
Returns the error description for the last executed operation

InitMIB ()
Inits the MIB subsystem

AddMIBDir (string dir)
Adds a MIB definition directory

array ReadMIB (string filename)
Reads a MIB file and returns an array describing the MIB

handle Open (string host, string community_name=”public”, snmp_version=”1”)
Establishes a SNMP connection with the given host. On error returns null.

array Read (array what, get_next=false)
Reads data described by *what* and returns it as an array

array Walk (string target, string root=“”)
Walks through SNMP objects

Open ()
Closes the SNMP connection

The following example was tested on a router running RouterOS. You need first to download the .mib file from your equipment manufacturer.

snmpexample.con

```
include SNMP.con

class Main {
   function Main() {
       var snmp=new SNMP();
       snmp.InitMIB();
       var data=snmp.ReadMIB("RouterOS.mib");
       // print mib data
       // echo snmp.ShowData(data);
       snmp.Open("192.168.2.1", "public");
       if (!snmp.Error) {
           // This prints all child objects of
               mtxrInterfaceStatsTable
           echo snmp.Walk("mtxrInterfaceStatsTable");
           echo snmp.Read(["DISMAN-EVENT-MIB::sysUpTimeInstance",
               "system.sysContact.0",
               "MIKROTIK-MIB::mtxrInterfaceStatsRxBytes.1"]);
           var err = snmp.Error;
           if (err)
               echo err;
       }
   }
}
```

Partial output:

```
Array {
        [0,"DISMAN-EVENT-MIB::sysUpTimeInstance"] => 226470600
        [1,"SNMPv2-MIB::sysContact.0"] =>
        [2,"MIKROTIK-MIB::mtxrInterfaceStatsRxBytes.1"] => 0
}
```

Both **Walk** and **Read** return key-value arrays.

Some manufacturers implement specific protocols. For example, Mikrotik, has a proprietary operating system called RouterOS on most of its equipments. Asides SNMP, it has some management APIs.

Concept Framework has a library called *standard.net.routeros* for managing RouterOS-based systems. It implements 4 static functions:

ROSConnect (user, password, host_ip, port=8728)
: Connects to the specified Router OS API on the given ip and logs in using user and password. On connection failed, returns -1. On success returns a socket handle

ROSDisconnect (socket)
Closes the RouterOS API connection

ROSQuery (sock, array command)
Executes the specified command on the given RouterOS sock. For a complete list of the supported commands, see the RouterOS wiki. On success, returns 0.

ROSResult (sock)
Returns the results of the previous query as an key-value array

routerosexample.con

```
import standard.net.routeros

class Main {
    Main() {
        var sock=ROSConnect("username", "passowrd", "192.168.2.1");
        if (sock>0) {
            if (!ROSQuery(sock, ["/interface/print"])) {
                // print the results
                echo ROSResult(sock);
            }
            ROSDisconnect(sock);
        }
    }
}
```

Replace *username* and *password* with your credentials. The output should be similar with:

```
Array {
    [0] =>
        Array {
            [0] => 0
            [1] => !re
            [2,".id"] => *1
            [3,"name"] => ether1-gateway
            [4,"default-name"] => ether1
            [5,"type"] => ether
            [6,"mtu"] => 1500
            [7,"l2mtu"] => 1598
            [8,"max-l2mtu"] => 2028
            [9,"mac-address"] => A4:8B:CD:CA:F1:EA
            [10,"fast-path"] => true
            [11,"rx-byte"] => 71641051794
            [12,"tx-byte"] => 25713436046
            [13,"rx-packet"] => 209931720
            [14,"tx-packet"] => 55925380
            [15,"rx-drop"] => 0
            [16,"tx-drop"] => 0
            [17,"rx-error"] => 0
            [18,"tx-error"] => 0
            [19,"running"] => true
            [20,"disabled"] => false
        }
[...]
    [7] =>
        Array {
            [0] => 1
            [1] => !done
        }
}
```

Note that except element [0] (the return code), all of the values are strings. The */interface/print* command will return an array describing all network interfaces on the RouterOS host.

standard.net.socket implements an ICMP(Internet Control Message Protocol) Ping function.

```
array Ping(string host, count=1, timeout_ms=5000, message_size=32)
```

Note, that the Ping function works only when running as a superuser.

ping.con

```
import standard.net.socket

class Main {
    Main() {
        echo Ping("www.google.com");
    }
}
```

Output:

```
Array {
       [0,"reply"] =>
              Array {
                     [0] =>
                            Array {
                                   [0,"from"] => 92.87.156.99
                                   [1,"bytes"] => 56
                                   [2,"time"] => 16
                                   [3,"TTL"] => 58
                            }
              }
       [1,"statistics"] =>
              Array {
                     [0,"from"] => 92.87.156.99
                     [1,"sent"] => 1
                     [2,"received"] => 1
                     [3,"lost"] => 0
                     [4,"statistics"] => 0
              }
}
```

Note that the result array is split in two sub-arrays, the “reply”, containing the reply-data and the “statistics” sub-array.

Concept Framework also provides support for the Netflow protocol. Netflow is a feature that was introduced on Cisco routers that give the ability to collect IP network traffic as it enters or exits an interface. By analyzing the data that is provided by Netflow a network administrator can determine things such as the source and destination of the traffic, class of service, and the cause of congestion. Netflow consists of three components: flow caching, Flow Collector, and Data Analyzer[11]. Various router manufactures are supporting this protocol, for example Cisco, Alcatel-Lucent, Hawei, Enterasys, and Mikrotik. Also, Linux, BSD and VMware servers support this feature.

The *standard.net.flow* import library implements two static functions:

```
array ParseNetflowPacket(string msg[, var templatehandle]);
DoneNetflowPacket(templatehandle);
```

[11] http://en.wikipedia.org/wiki/NetFlow on Febrary 12, 2014

ParseNetflowPacket will parse netflow v5 and v9 datagrams. For netflow v9 packets the *templatehandle* parameter is mandatory. Netflow v9 is based on templates, but the routers do not send the templates with each packets. *templatehandle* is used internally by the *ParseNetflowPacket* to keep a reference to a specific list of templates.

When using Netflow v9, a call to *DoneNetflowPacket*, if templatehandle is not null, is mandatory to avoid a memory leak.

The following example will listen on the UDP port 2055 for netflow packets.

```
include UDPSocket.con
import standard.net.flow

class Main {
    Main() {
        var t = new UDPSocket();
        if (t.Bind(2055)) {
            echo "Error in bind\n";
            return -1;
        }


        while (true) {
            try {
                var msg = t.Read(var ip, var port);
                var arr = ParseNetflowPacket(msg, var handle);
                echo arr;
            } catch (var exc) {
                echo exc;
                break;
            }
        }
        DoneNetflowPacket(handle);
        t.Close();
    }
}
```

Output:

```
Array {
    [0,"metadata"] =>
```

```
       Array {
           [0,"version"] => 5
           [1,"flows"] => 1
           [2,"uptime_ms"] => 275526500
           [3,"time_sec"] => 361926
           [4,"time_nanosec"] => 129450752
           [5,"engine_type"] => 0
           [6,"engine_id"] => 0
           [7,"sampling_interval"] => 0
    }
    [1,"flows"] =>
       Array {
        [0] =>
           Array {
            [0,"IPV4_SRC_ADDR"] => 192.168.2.101
            [1,"IPV4_DST_ADDR"] => 173.194.70.188
            [2,"IPV4_NEXT_HOP"] => 89.41.248.1
            [3,"INPUT_SNMP"] => 7
            [4,"OUTPUT_SNMP"] => 1
            [5,"IN_PKTS"] => 3
            [6,"IN_BYTES"] => 120
            [7,"FIRST_SWITCHED"] => 275496850
            [8,"LAST_SWITCHED"] => 275510110
            [9,"L4_SRC_PORT"] => 58929
            [10,"L4_DST_PORT"] => 5228
            [11,"pad1"] => 0
            [12,"TCP_FLAGS"] => 4
            [13,"PROTOCOL"] => 6
            [14,"SRC_TOS"] => 0
            [15,"SRC_AS"] => 0
            [16,"DST_AS"] => 0
            [17,"SRC_MASK"] => 0
            [18,"DST_MASK"] => 0
            [19,"pad2"] => 0
           }
       }
}
```

The *ParseNetflowPacket* function will parse both Netflow v5 and Netflow v9 datagrams.

11.12 Virtualization client

Concept Framework is able to interact with multiple virtualization systems by using *libvirt*[12]. libvirt is an open source API, daemon and management tool for managing platform virtualization. It can be used to manage Linux KVM, Xen, VMware ESX, QEMU and other virtualization technologies. These APIs are widely used in the orchestration layer of hypervisors in the development of a cloud-based solution[13]. *libvirt* functions are mapped via the *standard.lib.virt* import library.

According to the libvirt website, the following systems are supported:

- The KVM/QEMU Linux hypervisor
- The Xen hypervisor on Linux and Solaris hosts.
- The LXC Linux container system
- The OpenVZ Linux container system
- The User Mode Linux paravirtualized kernel
- The VirtualBox hypervisor
- The VMware ESX and GSX hypervisors
- The VMware Workstation and Player hypervisors
- The Microsoft Hyper-V hypervisor
- The IBM PowerVM hypervisor
- The Parallels hypervisor
- The Bhyve hypervisor
- Virtual networks using bridging, NAT, VEPA and VN-LINK.
- Storage on IDE/SCSI/USB disks, FibreChannel, LVM, iSCSI, NFS and filesystems

[12] See http://libvirt.org

[13] http://en.wikipedia.org/wiki/Libvirt on April 7th, 2014

For a simple example, connecting to a test service, consider the following example:

```
import standard.lib.virt

class Main {
    Main() {
        var conn = virConnect("test:///default");
        if (conn) {
            echo virInfo(conn);
            var arr=virListAllDomains(conn);
            echo arr;
            var id=arr[0]["uuid"];
            var d=virDomainOpenUUID(conn, id);
            // power on the first machine
            virDomainCreate(d);

            // show memory usage statistics
            echo virDomainMemoryStats(d);

            // create a snapshot
            virDomainSnapshotCreate(d);

            // list all saved snapshots
            var arr2=virDomainSnapshotList(d);
            for (var i=0;i<length arr2;i++)
                echo virDomainSnapshot(d, arr2[i]);

            // power off
            virDomainShutdown(d);
            // no longer referencing the domain
            virDomainDone(d);
            // close connection
            virClose(conn);
        } else
            echo "Error connecting";
    }
}
```

The output:

```
Array {
    [0,"type"] => Test
    [1,"version"] => 0.0.2
    [2,"model"] => i686
    [3,"memory"] => 3145728
    [4,"cpus"] => 16
    [5,"mhz"] => 1400
    [6,"nodes"] => 2
    [7,"sockets"] => 2
    [8,"cores/socket"] => 2
    [9,"threads/core"] => 2
} Array {
    [0] =>
       Array {
           [0,"name"] => test
           [1,"uuid"] => 6695eb01-f6a4-8304-79aa-97f2502e193f
           [2,"id"] => 1
           [3,"cpus"] => -1
           [4,"os"] => linux
           [5,"hostname"] =>
           [6,"maxmemory"] => 8388608
           [7,"maxvcpus"] => 2
           [8,"active"] => 1
           [9,"persistent"] => 1
           [10,"updated"] => 0
       }
}
```

standard.lib.virt implements the following functions: *virError, virConnect, virInfo, virClose, virNumOfActiveDomains, virNumOfInactiveDomains, virListAllDomains, virDomainOpen, virDomainOpenUUID, virDomainOpenName, virDomainDone, virDomainMemoryPeek, virDomainShutdown, virDomainSetAutostart, virDomainDestroy, virDomainSuspend, virDomainResume, virDomainUndefine, virDomainReboot, virDomainCreate, virDomainCreateXML, virDomainScreenshot, virDomainSendKey, virDomainOpenConsole, virStreamClose, virStreamRead, virStreamWrite, virDomainGetCPUStats, virDomainInterfaceStats, virDomainMemoryStats, virDomainBlockStats, virDomainSnapshotCreate, virDomain, virDomainHasCurrentSnapshot, virDomainSnapshotList, virDomainSnapshot, virDomainRevertToSnapshot, virDomainSnapshotDelete, virDomainSnapshotListChildren, virDomainManagedSave, virDomainManagedSaveRemove, virDomainSave, virDomainHasManagedSaveImage, virDomainGetJobInfo, virDomainAttachDevice, virDomainBlockJobAbort, virDomainMigrate,* *virConnectListStoragePools, virConnectListDefinedStoragePools, virConnectListNetworks, virConnectListDefinedNetworks, virNodeListDevices* and *virDebug*.

The documentation for all of these functions may be found on http://libvirt.org/, and will not be discussed in this book, the functions being simply wrappers to the native C functions. This means that a libvirt C function will be called from Concept using the same syntax.

However, Concept defines the following specific functions:

handle virConnect (connection_string[,string username][,string password][boolean read_only=false])
Connects to a given host, using optional username/password. If read_only is set to true, the client will not be able to modify any domain.
Connection strings may be:
QEMU:

```
qemu:///session          (local access to per-user instance)
qemu+unix:///session     (local access to per-user instance)
qemu:///system           (local access to system instance)
qemu+unix:///system      (local access to system instance)
qemu://example.com/system (remote access, TLS/x509)
qemu+tcp://example.com/system (remote, SASl/Kerberos)
qemu+ssh://root@example.com/system (remote, SSH tunnelled)
```

VMWare ESX:

```
vpx://example-vcenter.com/dc1/srv1
(VPX over HTTPS, select ESX server 'srv1' in datacenter 'dc1')
esx://example-esx.com
(ESX over HTTPS)
gsx://example-gsx.com
(GSX over HTTPS)
esx://example-esx.com/?transport=http
(ESX over HTTP)
esx://example-esx.com/?no_verify=1
(ESX over HTTPS, but doesn't verify the server's SSL
    certificate)
```

A complete list can be found on the *libvirt.org* website.

virClose (connection_handle)
Closes the given connection handle.

array virInfo (connection_handle)
Retrieves information about the current connection (see the previous example). The return is a key-value array, having the following keys: *type, version, model, memory, cpus, mhz, nodes, sockets, cores/socket* and *threads/core.*

virDomainDone (domain_handle)
Frees the local memory associated with a a domain handle, as returned by functions like *virDomainOpen, virDomainOpenUUID* or *virDomainOpenName.*

number virDomainStreamWrite (stream_handle, string buffer)
Writes buffer to a stream handle, as returned by *virDomainOpenConsole.* Returns the number of bytes written, or -1 in case of error.

number virDomainStreamRead (stream_handle, var buffer, number size)
Reads at most *size* bytes into buffer from a stream handle, as returned by *virDomainOpenConsole.*

virDomainStreamClose (stream_handle)
Closes a stream handle.

array virError ()
Returns a key-value array describing the error in the last performed operation. The used keys are: *code, domain, level, message, str1, str2, str3, int1* and *int2.* If no error occurred, it will return an empty array.

virDebug (boolean on)
Sets the debug flag to *on.* If set to true, error information will be printed to *stderr.*

array virDomain (domain_handle)
Returns an key-value array describing the domain. The keys are: *name, uuid, id, cpus, os, hostname, maxmemory, maxvcpus, active, persistent* and *updated.*

More information may be found the libvirt.org website.

11.13 OAuth 2.0 and social media

Concept Framework provides the **SocialAPI** class, defined in *SocialAPI.con* include file. This enables the development of Facebook, Google and LinkedIn applications.

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf[14].

The SocialAPI class contains the following members

SocialAPI (string appid, string appsecret)
Create a SocialAPI object using the given *appid* (or client id) and app secret (or client secret).

string FacebookLogin (string redirecturi, boolean encodeuri=true)
Returns the URL(as a string) to redirect the user for granting access to a Facebook application. Note that *redirecturi* must be set to an authorized address (in your Facebook app settings). If *encodeuri* is set to true, the URL will be encoded, for example *http://somedomain.com* will become *http%3A%2F%2Fsomedomain%2Ecom*. If the user will grant access, it will be redirected to the given *redirecturi*, with a *code* string parameter, received as a GET request, that it will be used in the call to *GetFacebookAccess*.

string GoogleLogin (string scope="profile email", string extra_parameters="", string redirecturi="http://localhost", encodeuri=true)
Returns the URL(as a string) to redirect the user for granting access to a Google application. Note that *redirecturi* must be set to an authorized address (in your Google app settings). If *encodeuri* is set to true, the URL will be encoded. The *scope* parameter, sets the authorization scope. For example, for accessing user profile and list of contacts, you may want to use *"profile email https://www.googleapis.com/auth/contacts.readonly"*. See Google

[14] http://tools.ietf.org/html/rfc6749 as of August 19th, 2014

API documentation for a complete list of scopes. *extra_parameters* may contain additional GET parameters, for in the format *"parameter1=somevalue&parameter2=2"*.

If the user will grant access, it will be redirected to the given *redirecturi*, with a *code* string parameter, received as a GET request, that it will be used in the call to *GetGoogleAccess*.

string LinkedInLogin (string scope="", string extra_parameters="", string redirecturi="http://localhost", encodeuri=true)
Returns the URL(as a string) to redirect the user for granting access to a LinkedIn application. Note that *redirecturi* must be set to an authorized address (in your LinkedIn app settings). If *encodeuri* is set to true, the URL will be encoded.

It is identical with *GoogleLogin* in regards of the parameters, returning instead a LinkedIn URL to redirect the user. For a complete list of the scopes, check the LinkedIn Developer Network.

boolean GetFacebookAccess (string code, string redirecturi="https://www.facebook.com/connect/login_success.html", encodeuri=true)
Exchanges the *code* received by the *redirecturi* indicated by the previous call to *FacebookLogin* with an *access token* needed for using the Facebook API. Note that *redirecturi* parameter must be identical with the one used in the call to *FacebookLogin*. If succeeded it returns **true**, on error, **false** and sets *LastError* accordingly.

boolean GetGoogleAccess (string code, string redirecturi="http://localhost")
Exchanges the *code* received by the *redirecturi* indicated by the previous call to *GoogleLogin* with an *access token* needed for using the Google API. Note that *redirecturi* parameter must be identical with the one used in the call to *GoogleLogin*. If succeeded it returns **true**, on error, **false** and sets *LastError* accordingly.

boolean GetLinkedInAccess (string code, string redirecturi="http://localhost")
Exchanges the *code* received by the *redirecturi* indicated by the previous call to *LinkedInLogin* with an *access token* needed for using the LinkedIn API. Note that *redirecturi* parameter must be identical with the one used in the call to *LinkedInLogin*. If succeeded it returns **true**, on error, **false** and sets *LastError* accordingly.

boolean GetTwitterAccess ()

Access a twitter app. Note, that no user interaction is needed for performing this call. Returns **true** if succeeded.

array FacebookAPI (string api, parameters=null, string method="GET", var plain_res=null)

Calls the given API, using *parameters* (a key-value array), using the given method (GET—POST—PUT—DELETE). If *plain_res* is given, it will hold the actual unparsed server response. The call will return a key-value array containing the requested data.

Note that this call may be made only after a successful call to GetFacebookAccess.

array GoogleAPI (string api, parameters=null, string method="GET", var plain_res=null)

Calls the given API, using *parameters* (a key-value array), using the given method (GET—POST—PUT—DELETE). If *plain_res* is given, it will hold the actual unparsed server response. The call will return a key-value array containing the requested data.

Note that this call may be made only after a successful call to GetGoogleAccess.

array LinkedInAPI (string api, parameters=null, string method="GET", var plain_res=null)

Calls the given API, using *parameters* (a key-value array), using the given method (GET—POST—PUT—DELETE). If *plain_res* is given, it will hold the actual unparsed server response. The call will return a key-value array containing the requested data.

array TwitterAPI (string api, parameters=null, string method="GET", var plain_res=null)

Calls the given API, using *parameters* (a key-value array), using the given method (GET—POST—PUT—DELETE). If *plain_res* is given, it will hold the actual unparsed server response. The call will return a key-value array containing the requested data. *Note that this call may be made only after a successful call to GetTwitterAccess.*

array FetchJSON (string url)

Fetches the given *url* using a maximum of 10 redirects and returns the data as an array (a parsed JSON).

string Fetch (string url)
: Fetches the given *url* using a maximum of 10 redirects and returns the data as a text buffer.

static array PicasaInfo (string username)
: Returns information about a given a *Picasa* user as a key-value array. Note that for this function no authorization is needed.

boolean InvalidateTwitterToken ()
: Invalidates a Twitter access token. Note that this function must be called after a successful call to *GetTwitterAccess*. Returns **true** if succeeded.

public string AccessToken (read/write)
: Holds the access token used for the API calls.

public string LastError (read/write)
: Holds the text of the last error.

public number Expires (read)
: Holds the session expire date as epoch time. If it is 0, the session will never expire or no successful authorization was performed.

The call flow is:

1. new SocialAPI(CLIENT_ID, CLIENT_SECRET)

2. Request redirect link for the user with SocialAPI.GoogleLogin(..redirecturi..)

3. Wait for a call to *redirecturi* with a code paramter (via GET method). Set code=WebVar("code")

4. Exchange the *code* with an *authorization token* via SocialAPI.GetGoogleAccess(code, *redirecturi*)

5. call SocialAPI.GoogleAPI("/v1/plus/me") or whatever API you're interested on.

A simple Concept UI application that allows in-app login to Facebook will look something like this:

```
include SocialAPI.con

include Application.con
include RImage.con
include RForm.con
include RTreeView.con
include RWebView.con

define FBAPI      "1463186XXXXXXXXX",
    "XXXXd16cXXXXXXXX8310XXXXXXXa77XX"

class LoginWindow extends RForm {
    public var WebView;

    LoginWindow(Owner) {
        super(Owner);
        WebView = new RWebView(this);
        WebView.Show();
    }
}

class MainForm extends RForm {
    var WebForm;
    var treeContacts;
    var fb;
    var image;
    var dummy_image;

    public function MainForm(Owner) {
        super(Owner);

        this.treeContacts = new RTreeView(this);
        this.treeContacts.Model = MODEL_LISTVIEW;
        this.treeContacts.AddColumn("", IMAGE_COLUMN);
        this.treeContacts.AddColumn("");
        this.treeContacts.AddColumn("");
        this.treeContacts.SearchColumn=1;
        this.treeContacts.Show();

        this.OnRealize=this.FBLogin;

        image = new RImage(null);
        dummy_image = new RImage(null);
        dummy_image.Filename="res/dummy.png";

        // create the social media object
        fb=new SocialAPI(FBAPI);
    }

    public function FBLogin(Sender, EventData) {
        WebForm = new LoginWindow(this);
        WebForm.Show();
        // get the social relocation address
        WebForm.OpenString("<body><a
            href='${fb.FacebookLogin("http://localhost")}'>Sign in
            with Facebook</s>");
        // intercept navigation in RWebView
        WebForm.WebView.OnNavigationRequested = this.OnRequest;
    }

    public function LoadUser(social, comments="") {
        var arr=social.FacebookAPI("/me");
        var name=arr["name"];

        if (name) {
            var target=dummy_image;
            var photo_link=social.FacebookAPI("/$user/picture",
                ["type" => "square", "redirect"=>"false"]);
            if (photo_link) {
                var data=photo_link["data"];
                if ((data) && (!data["is_silhouette"])) {
                    var url=data["url"];
                    var img_data=social.Fetch(url);
                    if (img_data) {
                        image.SetBuffer(img_data, "image.jpg");
                        target=image;
                    }
                }
            }
            // add the contact to a tree view
            this.treeContacts.AddItem([target, name, comments]);
        }
    }

    public function OnRequest(Sender, EventData) {
        if (Pos(EventData, "http://localhost")==1) {
            var arr=StrSplit(EventData, "code=");
            var code=""+StrSplit(""+arr[1], "#")[0];
            WebForm.Hide();
            // extract code and request access token
            if (fb.GetFacebookAccess(code, "http://localhost"))
                this.LoadUser(fb, "This is you");
            else
                CApplication.MessageBox(fb.LastError);
        } else
            // if not our redirect, allow navigation
            WebForm.WebView.URI=EventData;
    }
}

class Main {
    Main() {
        try {
            var Application=new CApplication(new MainForm(NULL));
            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            //echo Exception.Information;
        }
    }
}
```

Assuming that a valid FBAPI constant was defined, this application will extract the user photo and name and show them in a row in *treeContacts*.

The model is identical for Google and LinkedIn (just the *api* parameters are different).

The minimal Twitter application access example:

```
include SocialAPI.con

class Main {
    Main() {
        var g=new SocialAPI("m4XXXXXXXXXXXXXXXXXXXXXXX",
            "XXXXXXXXXXXA2XXXXXXXXXXXQXXXXXXXXXXA1DXXXXXXXXXXXX");
        if (g.GetTwitterAccess()) {
            echo
                g.TwitterAPI("/1.1/statuses/user_timeline.json?screen_name=charliesheen")
            // invalidate session
            g.InvalidateTwitterToken();
        }
    }
}
```

For advanced Twitter API, see documentation of the *Twitter* class (defined in *Twitter.con*).

11.14 Upgrading concept:// to TLS/SSLv3

Starting with version Concept Client 3.0 and Concept Server 3.0, concept:// and concepts:// protocols, are supporting TLS and SSLv3 sockets.

Note that when using the *concepts* secured protocol, the data transported by the TLS socket will be encrypted both by the secure socket and concept secured protocol. This is done purely for backwards compatibility and does not necessary mean a more secure method of data transfer. concepts protocol is similar with TLS in architecture, but it only uses RSA keys, instead of X.509 certificates[15], issued by trusted certificate authorities.

Each concept(s):// application must have its own certificate, using as domain the server application path. You can generate your own self-signed certificate, by using OpenSSL.

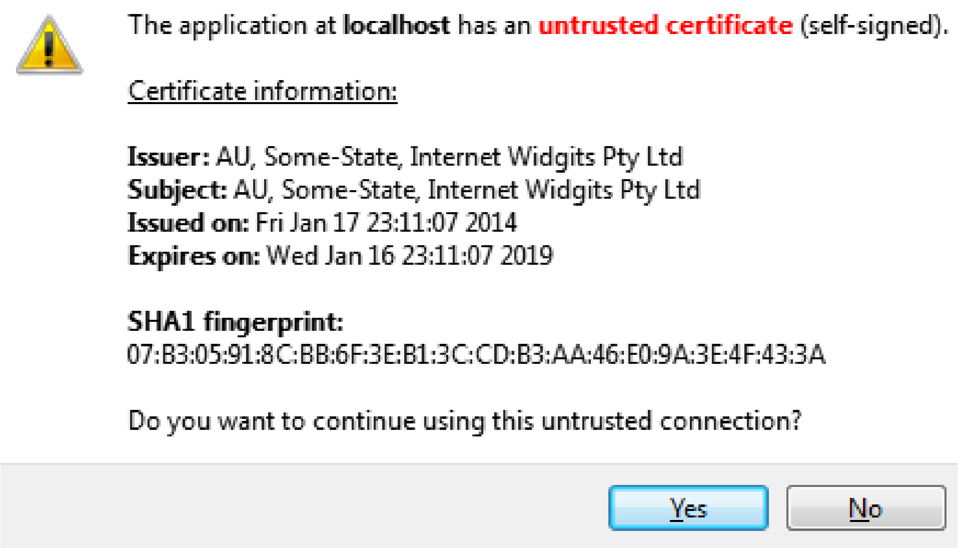
First, you must generate a root certificate authority, as it folows:

```
openssl genrsa -out rootCA.key 4096
openssl req -new -x509 -nodes -key rootCA.key -out rootCA.crt -days
    36500
```

The the actual certificate (using a 4096-bit key):

```
openssl genrsa -out privatekey.pem 4096
openssl req -new -key privatekey.pem -out publickey.cer
openssl x509 -req -in publickey.cer -CA rootCA.crt -CAkey
```

[15] In cryptography, X.509 is an ITU-T standard for a public key infrastructure (PKI) and Privilege Management Infrastructure (PMI). X.509 specifies, amongst other things, standard formats for public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm. (source: http://en.wikipedia.org/wiki/X.509 on September 6th, 2014)



The application at **localhost** has an **untrusted certificate** (self-signed).

Certificate information:

Issuer: AU, Some-State, Internet Widgits Pty Ltd
Subject: AU, Some-State, Internet Widgits Pty Ltd
Issued on: Fri Jan 17 23:11:07 2014
Expires on: Wed Jan 16 23:11:07 2019

SHA1 fingerprint:
07:B3:05:91:8C:BB:6F:3E:B1:3C:CD:B3:AA:46:E0:9A:3E:4F:43:3A

Do you want to continue using this untrusted connection?

Figure 11.3: Untrusted certificate prompt

```
    rootCA.key -CAcreateserial -out publickey.cer -days 365
```

For more documentation about *openssl*, check https://www.openssl.org.

This will offer you *untrusted* certificates, that will encrypt the data over the network, but the user will be notified about the untrusted certificate, as seen in figure 11.3.

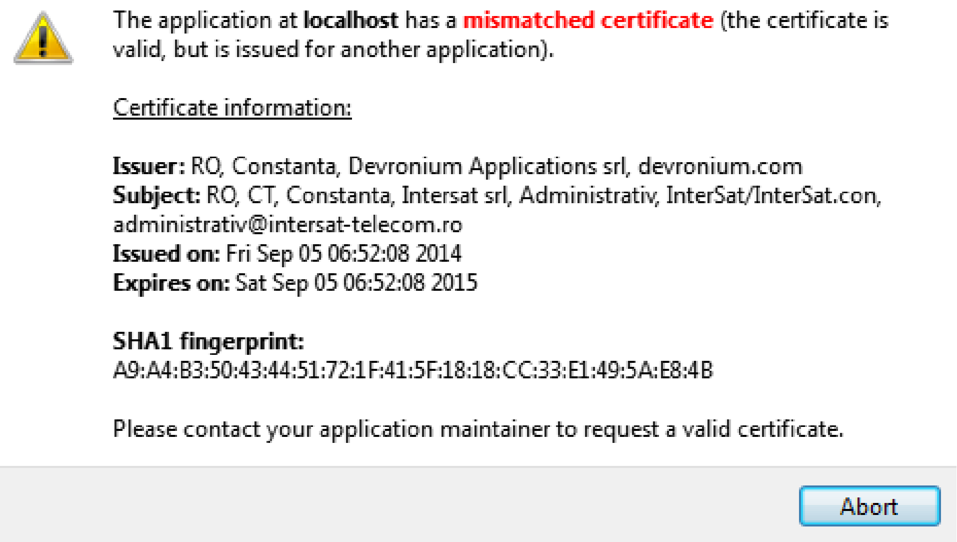
The actual connection must be upgraded in two steps. The first step consists of notifying the client to upgrade to TLS using *CApplication::Upgrade(mode)* API:

```
boolean Upgrade(mode=TLS_STRICT);
```

Where mode may be:

TLS_STRICT
This is the default option. Accepts only trusted and valid certificates, disabling the "Continue" option for the user as show in figure 11.4.

TLS_ACCEPT_UNTRUSTED



The application at **localhost** has a **mismatched certificate** (the certificate is valid, but is issued for another application).

Certificate information:

Issuer: RO, Constanta, Devronium Applications srl, devronium.com
Subject: RO, CT, Constanta, Intersat srl, Administrativ, InterSat/InterSat.con, administrativ@intersat-telecom.ro
Issued on: Fri Sep 05 06:52:08 2014
Expires on: Sat Sep 05 06:52:08 2015

SHA1 fingerprint:
A9:A4:B3:50:43:44:51:72:1F:41:5F:18:18:CC:33:E1:49:5A:E8:4B

Please contact your application maintainer to request a valid certificate.

Figure 11.4: Mismatched certificate prompt, with no continue option

Allows the user to choose if the untrusted, expired or mismatched certificate to be used, as show in figure 11.3.

TLS_SUPPORTED
Do not upgrade the connection, just check if is TLS/SSLv3 is supported.

CApplication::Upgrade(mode) returns *true* if TLS/SSLv3 is supported, or **false** if not.

The second step is to upgrade the server connection using the static *UpgradeTLS* API (defined in *win32.base.messages*):

```
number UpgradeTLS(string cert_file, string private_file);
```

cert_file is the public key (certificate) and *private_file* is the private key file. *UpgradeTLS* return 1 if succeeded, -2 if TLS/SSL is not supported on the server, -3 if the used certificate is invalid or 0 if the SSL engine or socked failed from any other reason.

You can get trusted certificates from lots of places, starting with Devronium Applications, the company developing Concept Application

Server to VeriSign.

The simplest concept:// TLS/SSLv3 application is show in the example below.

simple_tls_app.con

```
include Application.con
include RForm.con
include RLabel.con

class MyForm extends RForm {
    MyForm(parent) {
        super(parent);
        var infoLabel = new RLabel(this);
        switch (ConceptProtocolSecured()) {
            case 0:
                infoLabel.Caption = "concept:// protocol without TLS
                    is used";
                break;
            case 1:
                infoLabel.Caption = "concepts:// protocol without TLS
                    is used";
                break;
            case 2:
                labinfoLabelel.Caption = "concept:// protocol WITH
                    TLS is used";
                break;
            case 3:
                infoLabel.Caption = "concepts:// protocol WITH TLS is
                    used";
                break;
        }
        infoLabel.Show();
    }
}

class Main {
    function Main() {
            try {
                if (CApplication::Upgrade()) {
                    if (UpgradeTLS("./cert/publickey.cer",
                        "./cert/privatekey.pem")==1) {
                        var Application=new CApplication(new
                            MyForm(NULL));
                        Application.Init();
                        Application.Run();
                        Application.Done();
                    }
                } else
                    CApplication::Message("Please upgrade your
                        Concept Client");
            } catch (var Exception) {
                echo Exception;
            }
    }
}
```

Note concept protocol TLS upgrade may not be supported on some platforms.

Chapter 12

Multi-threading

Concept Core supports basic multi-threading. The core itself is optimized for speed, but if you need to share data between threads, the thread-safe version of the core should be used. On Microsoft Windows is located into the Concept/bin directory, and is named concept.core.X.Y.MT.dll, on other operating systems is usually in /usr/local/lib/libconceptcore.X.Y.MT (multiple files). For running all the applications on a server on the multi-threaded version, the standard versions of the concept core (the ones without the MT suffix) should be replaced with the MT version. For example, on MS Windows, you can replace concept.core.X.Y.dll with concept.core.X.Y.MT.dll(by renaming the MT version). Note that Concept Server must be shut down in order to allow you to replace the dynamic library.

For running only some applications on the MT core, you can create a .manifest file requesting the core to use the MT version. For example, if a program is called *test.con*, a file named *test.con.manifest* should be created in the same directory with *test.con*. The contents of the *.manifest* file should look like this:

test.con.manifest

```
[Application]
Multithreading =  1
```

For CLI programs, you can use the *-mt* parameter, for example:

```
concept -mt test.con
```

When explicitly running with the *-mt* parameter, no manifest is needed.

You should note that the MT version is slower than the non-threaded version due to use of semaphores in the Concept Core. A programmer can use its own semaphores, and then use the standard core version, having both speed and safe data sharing between threads, but this is not recommended, because of the explicit management of resources and variable access.

Note that all the threads run on the same process, and sharing variables between them is not an issue. Just be sure to use semaphores, to avoid consistency problems or writing violations. All the threading and semaphore functions are defined in *standard.lib.thread*.

Note that all threading examples should be run on the MT version of the Concept Core. It will run on the standard core version, but it may be prone to crash due to lack of variable access core semaphores.

Multi-threading may have a small negative impact of the JIT ability to run native code on the given machine, because the need for semaphores. However functions and loops using only local variables will be run with no problem.

In practice, I recommend you use background queues in external processes or a fixed number of reusable threads for doing the background work.

Note that some of the Concept static libraries may create background threads. For example, the *standard.net.opal* (SIP protocol) creates two or three concept threads. If you use this library, even if no threads are created from your application, the MT version of the core should be used.

12.1 Multi-threading and core limitations

Creating a thread is relatively straight forward. Concept has a simple, cross-platform threading mechanism based on only 4 functions:

RunThread (delegate thread_function, detached=false)
: Creates a thread and returns its identifier as a strictly positive number. On error will return 0 or a negative value. The detached flag is set to true, you can't use WaitThread on it. If detach is set to false, and you don't call WaitThread, a memory leak will occur.

WaitThread (number thread_identifier)
: Waits for the thread identified by *thread_identifier* to end. This function always return 0. Note that this function should not be used on detached threads. On Windows platforms it will work, but on POSIX threads will not.

KillThread (number thread_identifier)
: Forcefully ends a thread. This function should not be used, because it may generate unpredictable results. The correct way of ending a thread is by exiting the thread function via a return statement.

Sleep (number milliseconds)
: Causes the current thread to sleep for the given milliseconds.

On the Concept MT Core, all member variables use an internal semaphore for access, in order to avoid concurrent writes. This has some speed penalty, but the core remains reasonable fast. When the core detects that the thread count is 1 (only the main thread), it doesn't uses the semaphore anymore, being only a little slower than the standard non-threaded version.

Most of the Concept Framework is thread safe, however, you should **avoid setting or getting properties for the UI objects outside the main thread**. It will work, but it can generate some inconsistency with the objects. For example, two threads may simultaneously read the same property, but reading the property produces some consequences, for example, when getting an image from an web cam. Is difficult to know which frame to which thread will end up.

For every non-detached thread (default), a corresponding call to WaitThread must be made.

A simple multi-threading program:

```
import standard.lib.thread

class Main {
   foo() {
       for (var i=0; i<10 ;i++)
           echo "$i;";
   }

   Main() {
       var[] threads;
       for (var i=0; i<10; i++)
           threads[i]=RunThread(this.foo);

       // allow the threads to finish
       for (i=0; i<8; i++)
           WaitThread(threads[i]);
   }
}
```

The output:

```
0;1;2;3;0;0;0;0;0;0;0;0;0;4;1;1;1;1;1;1;1;1;1;5;2;2;2;2;2;
2;2;2;2;6;3;3;3;3;3;3;3;3;3;7;4;4;4;4;4;4;4;4;4;8;5;5;5;5;
5;5;5;5;5;9;6;6;6;6;6;6;6;6;6;7;7;7;8;9;7;7;7;7;8;8;7;7;8;
8;8;8;9;9;8;8;9;9;9;9;9;9;
```

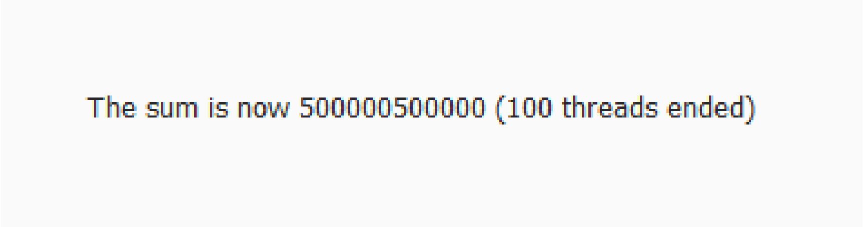
The threads should be used only for background computations. Also, note that most database drivers are thread safe, but may generate unpredictable results when simultaneous queries are made from various threads, so you should always make your data queries from the same thread or use semaphores to avoid simultaneously sending multiple commands on the same connection.

You could use the InterApp messaging system to synchronize your threads with the main UI thread. Just send messages to the same application, by using SendMessage(GetAPID(), 101, ”Thread finished”). Remember that InterApp only works for concept:// application (it doesn’t work for console applications).

The fullowing example computes the sum of the first 1,000,000 natural numbersby using 100 threads. Each thread computes the sum for a set of 10,000 numbers. This example must be run via CAS root and Concept Client, with the Concept MT core.

multithreadingexample.con

```
import standard.lib.thread
include Application.con
include RLabel.con
import standard.lib.thread

class ThreadData {
    var start;
    var end;

    ThreadData(start, end) {
        this.start = start;
        this.end = end;
    }

    Run() {
        // get current APID
        var apid = GetAPID();
        var result = 0;
        for (var i=start; i<end; i++)
            result += i;

        // send result as a string
        SendAPMessage(apid, 1, ""+result);
    }
}

class MainForm extends RForm {
    var label;
    // add the last number (note i<1000000)
    // this means that 1000000 will not be actually
    // added
    var sum = 1000000;
    var threads = 0;

    MainForm(owner) {
        super(owner);
        label = new RLabel(this);
        label.Caption = "Computing sum(0,1,..,1000000)";
        label.Show();

        for (var i = 0; i<1000000 ; i+=10000) {
            var t = new ThreadData(i, i+10000);
            // don’t wait for the thread
```



The sum is now 500000500000 (100 threads ended)

Figure 12.1:

```
            RunThread(t.Run, true);
        }
    }


    OnInterAppMessage(SenderAPID, MSGID, Data) {
        if (MSGID == 1) {
            threads++;
            sum += value Data;
            label.Caption = "The sum is now $sum ($threads threads
                ended)";
        }
    }
}

class Main {
    Main() {
        try {
            var mainform=new MainForm(null);
            var Application=new CApplication(mainform);
            Application.ShellDebug=mainform.OnInterAppMessage;
            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            CApplication::Message(Exception, "Uncaught exception",
                MESSAGE_ERROR);
        }
    }
}
```

The output is shown in figure 12.1.

Note that as per-CPU core basis, is more efficient to process all the data in a single thread than to compute in multiple threads. The multi-thread computation approach is somehow efficient on multi-core/multi-CPU systems, when the number of threads is somehow less or equal than the core count.

The previous example is highly inefficient, because the actual thread creations and APID message routing time surpass the time needed for adding 10,000 natural numbers.

As a conclusion, you should only use threads where is really needed. Threads are great for background computations or in server applications, for managing client communication. However, for basic UI applications there is rarely a need for multiple threads. Unless you're creating a game, a server, a computation intensive application, real-time communications like VoIP, or you need to create child processes that need to be monitored, you should not need to create a thread.

12.2 Semaphores

When using shared variables between threads, a semaphoring mechanism is included in Concept Framework, for ensuring read/write consistency. This mechanism is additional to the Concept Core internal semaphores.

A semaphore has two operations: P and V. The P operation wastes time or sleeps until a resource protected by the semaphore becomes available, at which time the resource is immediately claimed. The V operation is the inverse: it makes a resource available again after the process has finished using it.

You could see P as a lock attempt and V as a release. Is important that each P operations has a corresponding V operation in the same thread.

Semaphores are defined in *standard.lib.thread* import library.

semcreate ()

Creates a new semaphore. Returns the semaphore handle as a number.

semdone (number semaphore_handle)

Destroys a semaphore.

seminit (number semaphore_handle, number val)
Initializes a semaphore to the given value

semp (number semaphore_handle)
Performs a P operation on the semaphore (tries a lock or waits until the resource becomes available)

semv (number semaphore_handle)
Performs a V operation on the semaphore (releases the lock on the resource, making the resource avialable)

Alternatively you could use the simpler class ReentrantLock:

Lock ()
Tries a lock or waits until the resource becomes available

Unlock ()
Tries a lock or waits until the resource becomes available

The lockings are useful when you must ensure that a variable value won't change in a critical step. For example, when adding a constant to a variable, first the variable value is read and then the sum is performed, and then, the result is stored back to the variable. This is a critical step, since another thread can modify the value between the add operation and the final assignment, causing inconsistency.

```
include ReentrantLock.con

class Main {
    var semaphore;
    var sum;

    foo() {
        // ensure sum won't be changed
        semaphore.Lock();
        sum = sum + 1000;
        semaphore.Unlock();
    }

    Main() {
        semaphore = new ReentrantLock();
        var[] threads;
        for (var i=0 ;i<10; i++)
            threads[i]=RunThread(this.foo);

        // allow the threads to finish
        for (i=0; i<8; i++)
            WaitThread(threads[i]);
        echo sum;
    }
}
```

When running multiple threads, the *ReentrantLock* will be almost surely necessary to ensure the data in correctly synchronized. Note that when using the InterApp message subsystem, semaphores are not needed, because the message received callback is run in a loop in the main thread.

The same example can be rewritten to use the low level APIs.

```
import standard.lib.thread

class Main {
    var semaphore;
    var sum;

    foo() {
        // ensure sum won't be changed
        semp(semaphore)
        sum = sum + 1000;
        semv(semaphore)
    }

    Main() {
        semaphore = semcreate();
        seminit(semaphore, 1);
        var[] threads;
        for (var i=0 ;i<10; i++)
            threads[i]=RunThread(this.foo);

        // allow the threads to finish
        for (i=0; i<8; i++)
            WaitThread(threads[i]);
        echo sum;
        semdone(semaphore);
    }
}
```

However, the *ReentrantLock* class is recommended.

12.3 Green threads

Beside native threads, Concept also supports green threads. These are light weight (minimum overhead) threads that are scheduled by the Concept Core instead of the operating systems. These threads have many advantages and/or disadvantages. Note that this kind of threads may be used safely under the standard core, not needing the multi-threaded version of Concept.

The **advantages** are:

Creation time
Hundreds of thousands green thread may be created in up to a couple of seconds

No need for semaphores
They are scheduled by the Concept Core, there is no need for semaphores, the Core ensuring that each and every operation is atomic.

Minimal memory overhead
Green threads use only a few bytes for each thread. For example, 100,000 green threads use about 17MB of memory on a 32-bit system and up to 22MB on a 64-bit server.

Faster than native threads
Concept uses internal semaphores to ensure consistency when running native threads. This is making the MT core slower than the standard core. Green threads, not needing synchronization, run without any problems on the standard core, making them significantly faster.

The **disadvantages** are:

Using just one CPU core
: Green threads are running using just one core (unlike native threads that are using all the available cores). This may be solved by using by mixing native threads with green threads.

All threads may block if a single thread is blocking
: Special attention is required for blocking operations, for example socket read. If a socket is performing a blocking read (waiting for data) in a green thread, all the threads in the given queue will block. Blocking operations should be avoided in this case.

Green threads are logic threads, without no physical native threads behind. As with any APIs it should be used for the right problem, for example, strategy games, in which each gaming entity will have a corresponding green thread.

The *Greenloop* class, defined in *Greenloop.con* provides APIs for creating and running green threads. It implements the following members:

Greenloop

number Run(array delegates)
: Runs the given *delegates* (as an array) in a green thread loop. It returns 0 on success, non-zero on error.

number Add(delegate thread)
: Adds a *thread* to a queue while the loop is running. It returns 0 on success, or non-zero on error.

static number Go(array delegates)
: Static version, of run (avoiding creating a new object).

Note that all green threads must catch their exceptions. An uncaught exception will cause loop termination.

The following example will create three green threads: **greenHello.con**

```
include Greenloop.con

class Main {
   Green1() {
      for (var i = 0; i < 3; i++)
```

```
            echo "${@member} iteration $i\n";
    }

    Green2() {
        for (var i = 0; i < 3; i++)
            echo "${@member} iteration $i\n";
    }

    Green3() {
        for (var i = 0; i < 3; i++)
            echo "${@member} iteration $i\n";
    }

    Main() {
        var g=new Greenloop();
        if (g.Run([this.Green1, this.Green2, this.Green3]))
            echo "Error creating green loop";
    }
}
```

The output:

```
Green1 iteration 0
Green2 iteration 0
Green3 iteration 0
Green1 iteration 1
Green2 iteration 1
Green3 iteration 1
Green1 iteration 2
Green2 iteration 2
Green3 iteration 2
```

Note that the *Main* function could be replaced with the code bellow, resulting in the same output.

```
    Main() {
        if (Greenloop::Go([this.Green1, this.Green2, this.Green3]))
            echo "Error creating green loop";
    }
```

A green thread may add additional threads by using the *Add* method.

greenHello2.con

```
include Greenloop.con

class Main {
    var g;

    Green1() {
        // dynamically create a thread (Green3)
        g.Add(this.Green3);

        for (var i = 0; i < 3; i++)
            echo "${@member} iteration $i\n";
    }

    Green2() {
        for (var i = 0; i < 3; i++)
            echo "${@member} iteration $i\n";
    }

    Green3() {
        for (var i = 0; i < 3; i++)
            echo "${@member} iteration $i\n";
    }

    Main() {
        g=new Greenloop();
        if (g.Run([this.Green1, this.Green2]))
            echo "Error creating green loop";
    }
}
```

The output:

```
Green1 iteration 0
Green3 iteration 0
Green2 iteration 0
Green1 iteration 1
Green3 iteration 1
Green2 iteration 1
Green1 iteration 2
Green3 iteration 2
Green2 iteration 2
```

Note the slightly different execution order (*Green1, Green3, Green2*). When a thread is added, it will be queued just after the parent thread.

Green threads are compiled by the JIT core at the first run, for performance reasons. In some cases, if no JIT exit condition is reached, for example, a function call, member access or an echo call, an explicit JIT exist must be performed.

For example:

badGreen.con

```
include Greenloop.con

class Main {
    Green1() {
        var i = 1;
        while (i) {
            i++;
        }
    }

    Green2() {
        echo "This is never called";
    }

    Main() {
        if (Greenloop::Go([this.Green1, this.Green2]))
            echo "Error creating green loop";
    }
}
```

This will run forever, and *Green1* will never give a chance to *Green2* to execute. This can be fixed by adding an explicit call to *yield* (a dummy function).

```
include Greenloop.con

class Main {
    Green1() {
        var i = 1;
        while (i) {
            i++;
            // Give a chance to other threads to execute
            yield();
        }
    }

    Green2() {
        echo "This gets called eventually";
    }

    Main() {
        if (Greenloop::Go([this.Green1, this.Green2]))
            echo "Error creating green loop";
    }
}
```

Will output:

```
This gets called eventually
```

Note that it is not mandatory to call yield, unless there is no function call or object member access.

If *Green1* is defined as:

```
    Green1() {
        var i = 1;
        while (i) {
            // dummy operation
            i += sin(i);
        }
    }
```

No call to *yield* is needed, because *sin* will actually provide the core the opportunity to switch to the next thread.

In practice, the context switching is made when a JIT exit is encountered, or a loop structure ends (*for*, *do..while* or *while*) for performance reasons.

12.4 Workers

Another alternative to native threads, is the Concept Worker. This special native threads run in a different application context, inside the same process. A worker cannot access the application run-time objects, to avoid the synchronization overhead. Instead, it uses a message exchange system. When green threads cannot be used (eg. blocking calls), workers are a better idea than native threads. Note that for worker threads, you can use the standard concept core (no need for using the multi-threaded core).

For creating a **Worker** you need to first include *Profiler.con*.

Worker (string classname, string parameter = “”)
Constructs a Worker, creating a object of *classname* using the given *parameter*. Parameter is useful for initialization data. If it is not needed, should be set to an empty string

number IsActive ()
Returns true if the worker is active, false if it ended or -1 in case of error.

Join ()
Waits for the current worker to finish.

Exit ()
Forcefully ends a thread. Note, this function should be avoided, because it may cause memory leaks.

number AddData (string)
Adds data to the worker queue. This function should be used by the host application to send data to the worker. Returns the number of items in the queue, including the current one.

number GetResult (var string result)
Gets the data from the worker queue. This function should be used by the host application to get the results from the worker. Returns the number of items in the queue, including the current one. If no data is available, it will return 0. If the return is non-zero, *result* will contain the data from the worker.

static number Pending (var string data)

Gets the next data in queue. This function should be called by the worker. It returns the number of queued elements. If the return is non-zero, it will set *data* to the given string buffer (set by the application via *AddData*)

static number Result (var string result)
Adds a result in the result queue. This function should be called by the worker. It returns the number of queued results. The application will read the result by using the *GetResult* function.

Note that all *Get** functions will remove data from the queue. Also, the Worker destructor will call *Join*, so be sure to notify the Worker when the applications is done.

The following example will create the simplest worker:
simpleWorker.con

```
include Worker.con

class A {
    A(x) {
        // x is "Hello world !";
        echo "$x\n";
        while (true) {
            if (Worker::Pending(var data)) {
                if (data == "Done") {
                    echo "Bye !";
                    return;
                }
                echo "RECEIVED: $data\n";
                // set the result
                Worker::Result("ECHO $data\n");
            } else {
                // give the CPU a chance to do something else
                Sleep(1);
            }
        }
    }
}

class Main {
    Main() {
        // create
        var w = new Worker("A", "Hello world !");
        for (var i = 0; i < 10; i++) {
            w.AddData("Hello $i !");
            while (!(var idx = w.GetResult(var data)))
                Sleep(1);
            echo "$idx: " + data;
        }
        w.AddData("Done");
    }
}
```

The output:

```
Hello world!
Worker received: Hello 0 !
Worker replied: Hello to you too!
Worker received: Hello 1 !
Worker replied: Hello to you too!
Worker received: Hello 2 !
Worker replied: Hello to you too!
Worker received: Hello 3 !
Worker replied: Hello to you too!
Worker received: Hello 4 !
Worker replied: Hello to you too!
Worker received: Hello 5 !
Worker replied: Hello to you too!
Worker received: Hello 6 !
Worker replied: Hello to you too!
Worker received: Hello 7 !
Worker replied: Hello to you too!
Worker received: Hello 8 !
Worker replied: Hello to you too!
Worker received: Hello 9 !
Worker replied: Hello to you too!
Bye !
```

Using a hybrid application with both green threads and workers, eliminates completely the need for the native threads.

Remember that data cannot be directly shared between workers and parent thread. For complex data structures serialization may be used, as shown in the next example:

advancedWorker.con

```
include Worker.con
include Serializable.con

define OPERATION_BYE  1
define OPERATION_ADD  2
define OPERATION_DONE 3

class Message extends Serializable {
    var op;
    var data;

    Message(data = "", op = OPERATION_ADD) {
        this.data = data;
        this.op = op;
    }
}

class Student {
    var Name;
    var Grade;

    Student(Name = "", Grade = "") {
        this.Name = Name;
        this.Grade = Grade;
    }
}

class ChildWorker {
    WriteStudentToDatabase(student) {
        // pretend we write to the database
        echo "INSERT INTO student(name, grade) VALUES
            ('${student.Name}', '${student.Grade}');\n";
    }

    ChildWorker(init) {
        while (true) {
            // wait for a message
            if (Worker::Pending(var data)) {
                var msg = Message::UnSerialize(data, true);
                if (!msg)
                    break;

                switch (msg.op) {
                    case OPERATION_BYE:
                        // exit this worker
                        return;
                    case OPERATION_ADD:
                        WriteStudentToDatabase(msg.data);
                        msg.op = OPERATION_DONE;
                        Worker::Result(msg.Serialize(""));
                        break;
                }
            } else
                Sleep(1);
        }
    }
}

class Main {
    Main() {
        var w = new Worker("ChildWorker");
        w.AddData((new Message(new Student("Michael",
            "A+"))).Serialize(""));
        w.AddData((new Message(new Student("Andrew",
            "B-"))).Serialize(""));
        w.AddData((new Message(new Student("John",
            "A"))).Serialize(""));

        // wait for 3 messages
        var messages = 3;
        while ((w.IsActive()) && (messages)) {
            if (w.GetResult(var result)) {
                messages--;
                var msg = Message::UnSerialize(result, true);
                if (msg.op == OPERATION_DONE)
                    echo "Successfully written ${msg.data.Name} into
                        the database\n";
            }
        }
        w.AddData((new Message("", OPERATION_BYE)).Serialize(""));
        w.Join();
    }
}
```

The output:

```
INSERT INTO student(name, grade) VALUES ('Michael', 'A+');
INSERT INTO student(name, grade) VALUES ('Andrew', 'B-');
Successfully written Michael into the database
INSERT INTO student(name, grade) VALUES ('John', 'A');
Successfully written Andrew into the database
Successfully written John into the database
```

Avoid creating a number of workers greater than number of CPU cores times number of threads per core on the machine. Creating too many threads on any machine, will make your application run slower. When a large number of threads is needed, I recommend using a few workers and lots of green threads.

The use of workers is strongly encouraged when dealing with blocking calls. Keep in mind that workers are lightweight, and have no speed penalty, in contrast with native threads, which will run at about 3/4, and possibly even slower for recursive function calls, compared to the single thread concept core.

12.5 Parallel computing using GPU

Concept is able to run code in GPU's. A GPU(graphics processing unit) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display. Modern GPUs are very efficient at manipulating computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel. In a personal computer, a GPU can be present on a video card, or it can be on the motherboard or-in certain CPUs-on the CPU die[1].

A GPU high performance in parallel computations, makes it ideal for a set of problems that can be parallelized, especially in sorting, AI or NP-complete problems. It is important to chooses to use parallel computation when is actually needed and some significant performance increase can be gain. As a note, the GPUs usually use a different memory space than CPUs, resulting frequent data copy between CPU memory and GPU memory.

[1] http://en.wikipedia.org/wiki/Graphics_processing_unit, March 17, 2014

Concept provides the *Parallel* class, defined in *Parallel.con* for handling parallel computations. Note that not every Concept function may be parallelized. A function (called entry-point function) must meet the following conditions in order to qualify for parallelization:

1. A function must create no class objects.
2. If a function creates an internal array, it must statically refer the last element of the array, in order for the compiler to statically create an array of the required size.
3. It is recommended that all function parameters are declared by type. If not, Concept Core will try to guess the parameters type.
4. Non-homogeneous arrays are not permitted. Only static strings and number arrays are allowed. Matrices must be transformed into linear vectors.
5. Parameters must be either in or out. It cannot be both. All out parameters, regardless of their type, must be declared with *var*.
6. Operators like *typeof*, *classof* and *value* are not allowed.
7. A function may refer only members of the current class (*this*). The referenced member must only return *number* values.
8. Recursion is not allowed.
9. A function may call another function only using number parameters and bridge arrays. A bridge array is an array that was given as a parameter from the entry-point function (the first function parallelized).
10. Except math static functions, *ord* and *chr*, no static function call is allowed.
11. The entry-point function return value is ignored. However, a function called by the entry point function may return a number value.
12. Real numbers are limited to *float* (32 bit), instead of Concept double precision (64 bit).
13. Division by zero will not generate a run-time error.

14. *try/throw/catch* statements are ignored and should not be used.

15. string and array concatenation is not allowed.

16. the [ ] operator, when used both in strings and array may reference just one character, respectively just one number. Expressions like *str[1] = “abc”* are not allowed.

17. *echo* will not work on NVIDIA GPUs.

18. Every entry-point function must have the first parameter defined as a number, must not be a reference. The actual function parameters must follow this parameter.

As a note, it is not mandatory to run the parallel computing application as multi-threaded.

I’ve implemented this interface to be trivial, and easy to use and understand by any developer. The parallel interface is implemented over OpenCL (but could be changed to Nvidia CUDA APIs without changing the Concept APIs).

Consider the following example:

```
include Parallel.con
import standard.lib.str

class Main {
    foo(workindex, string in, var string out) {
        out[workindex] = chr(ord(in[workindex]) + 1);
    }

    Main() {
        try {
            var str = "Hello World!";
            var res = __gpu (foo, [str, str]);
            // output parameter is on the second position
            echo res[1];
        } catch (var exc) {
            echo exc;
        }
    }
}
```

You probably noted the **__gpu** call. This is a macro (identical with **__parallel**), that will simply call:

```
define __gpu  Parallel::Go
define __parallel Parallel::Go

Parallel::Go(delegate d, array parameters);
```

This is the synchronous interface of the parallel computation APIs. The *d* parameter is the function that needs to be parallelized and run, and the parameters are the function parameters (note that the *workindex* parameter must be skipped). For output parameters (in this case *out*), only an array of the required size must be given, regardless of its contents. This function will return an array, containing the output values. In our example, the out string is on the second position of the array, because is the second parameter in the function call.

The same function can be run asynchronously by using:

```
include Parallel.con

class Main {
    foo(workindex, string in, var string out) {
        // for parallel code, we can skip chr/ord for
        // strings. In parallel mode all strings
        // are regarded as number arrays
        out[workindex] = in[workindex] + 1;
    }

    Main() {
        try {
            var str = "Hello World!";

            var[] res;
            var p = new Parallel();
            if (p.Use(foo, var err)) {
                p.Run([arr, arr]);
                DoSomeTimeConsumingStuff();
                // get the results from the last call
                res = p.Join();
                echo res[1];
                p.Unload();
            }
        } catch (var exc) {
            echo exc;
        }
    }
}
```

For understanding the *workindex* parameter, let's see the same code written as standard code:

```
include Parallel.con
import standard.lib.str

class Main {
    foo(workindex, string in, var string out) {
        out[workindex] = chr(ord(in[workindex]) + 1);
    }

    Main() {
        var str = "Hello World!";
        var out = str;
        for (var workindex = 0; workindex < length str; workindex++)
            foo(workindex, str, out);
        echo out;
    }
}
```

workindex is simply the iteration, int the last example, and in the previous example, is the parallel thread and/or iteration number.

In either cases, the output is:

```
Ifmmp!Xpsme"
```

A parallel function may also call functions defined in the current class:

```
include Parallel.con
import standard.C.time
import standard.C.math

class Main {
    var member=1;

   foo2(array in, i) {
       return this.foo3(in, i);
   }

   foo3(array in, i) {
       return sqrt(in[i]);
   }

   foo(workindex, array in, var out) {
       // you can even reference members in the
       // current class
       // Note that this.member value will be linked
       // statically at compile time. It its value
       // will change, it won't be reflected in the GPU
       var test = [0, this.member, 2, 3, 4];
       for (var i=1; i<100000; i++)
           out[workindex] += sin(foo2(in, test[3]));
   }

   Main() {
       try {
           var arr = new [];
           // ensure that we have 1000 elements in the array
           // this step is extremely important, otherwise
           // the core will not be able to estimate the
           // array length
           arr[999] = 0;
           var result = __gpu (foo, [arr, arr]);
           var output = result[1];
           echo output;
       } catch (var exc) {
           echo exc;
       }
   }
}
```

The __gpu calls will recompile the Concept byte code into OpenCL source or byte code. This way, a programmer will use only one programming language. You may also use functions like *get_global_id, get_local_id, get_global_size, get_local_size*, get_work_dim, get_num_groups, get_group_id or *barrier*. See OpenCL documentation for a complete list of functions.

Chapter 13

Media, voice over IP and telephony

Concept Frameworks provides import libraries for image processing like: *win32.graph.freeimage* (portable, not related to MS Windows), *standard.graph.svg* (SVG support), *standard.graph.svgt* (SVG tiny support), *standard.graph.imagemagick* (multiple image formats supoport), *standard.lib.poppler* (PDF support), *standard.lib.face* (face detection) and *standard.lib.ocr* (Optical Character Recognition).

About two years ago I've started an Internet telephony project, and decided to put all the reusable code I've used into Concept Framework. Then, I decided to add support for high-compression codecs like Speex and later Opus, and SIP protocol support via *standard.net.opal* or OPALSIP.con. Speex and Opus are supported both on the Concept Client and CAS. This makes CAS a great solution for real-time communications.

The mobile version of Concept Client is highly optimized for VoIP, enabling jitter-free calls on the concept:// protocol, with optional UDP support. Also, basic DRM (Digital Rights Management) is supported for VoIP or audio applications.

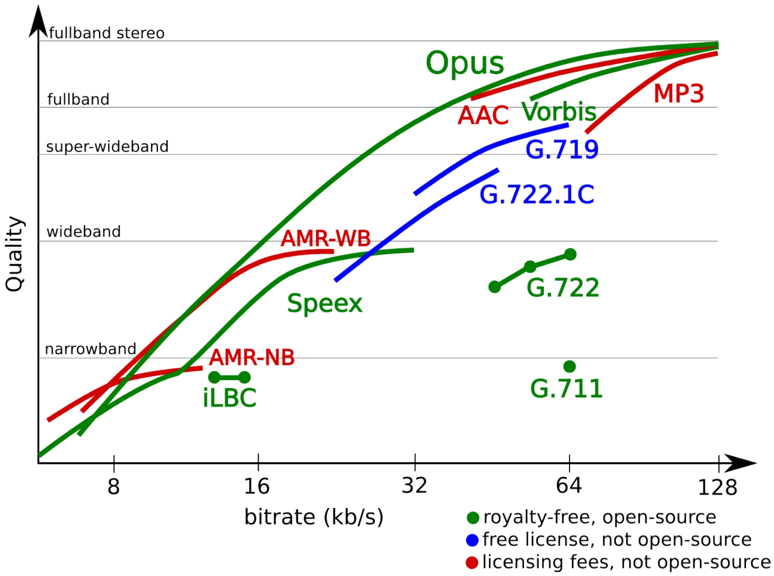


Figure 13.1:
OPUS codec comparison

13.1 Codecs

Concept supports two highly optimized codecs: Speex and Opus.

Speex is an Open Source/Free Software patent-free audio compression format designed for speech. There are lots of VoIP applications, servers and hardware supporting Speex. However, Speex is obsoleted by Opus.

Opus is a totally open, royalty-free, highly versatile audio codec. Opus is unmatched for interactive speech and music transmission over the Internet, but is also intended for storage and streaming applications. It is standardized by the Internet Engineering Task Force (IETF) as RFC 6716. Opus can handle a wide range of audio applications, including Voice over IP, videoconferencing, in-game chat, and even remote live music performances. It can scale from low bit-rate narrowband speech to very high quality stereo music. See figure 13.1[1].

Note that not all the codecs are supported with every client version. You

[1] Source: http://opus-codec.org/comparison/ as shown on January 23, 2014

can check the supported codecs by querying the client via *CApplication.Query*:

```
var codecs=CApplication.Query("Codecs", false);
```

If *codecs* is empty, it means that only Speex is supported (desktop version). On mobile version, codecs are returned as a string, separated by ";". For example:

```
Speex;Opus;DRM
```

All Concept Client mobile version support both Speex and Opus. Some desktop versions support only Speex (but recent version will also support Opus and DRM).

The remote audio is managed via the *OCV/RAudioStream.con* class (will be discussed in the next section).

On the CAS a set of four classes manage the compression/decompression of voice packages. For Speex, the SpeexEncoder and SpeexDecoder, both defined in SpeexEncoded.con, using the following interfaces:

SpeexEncoder:

BitsHandle : number property
Gets the bits handle to be used with the low-level APIs

StateHandle : number property
Gets the state handle to be used with the low-level APIs

SampleRate : number property
Sets the sample rate in Hz (default 16000)

FrameSize : number property (read-only)
Gets the Speex frame size

Encode (string buffer, var chunk_size=null)
Encodes the buffer. Chunk size is the Speex encoded chunk size. Returns the encoded Speex buffer.

Reset ()
Resets the internal audio buffer

SpeexEncoder:

BitsHandle : number property
Gets the bits handle to be used with the low-level APIs

StateHandle : number property
Gets the state handle to be used with the low-level APIs

SampleRate : number property
Sets the sample rate in Hz (default 16000)

Quality : number property
Sets the encoder quality, with 0 the lowest, 10 the highest. The output buffer will use more bytes with higher quality. Default is 5 (medium-low quality). A value of 8 should provide a balance between quality and used bandwidth.

FrameSize : number property (read-only)
Gets the Speex frame size

Encode (string buffer, var chunk_size=null)
Encodes the buffer. Chunk size will be set to the Speex encoded chunk size. Returns the encoded Speex buffer.

Reset ()
Resets the internal audio buffer

SpeexDecoder:

BitsHandle : number property
Gets the bits handle to be used with the low-level APIs

StateHandle : number property
Gets the state handle to be used with the low-level APIs

SampleRate : number property
Sets the sample rate in Hz (default 16000)

FrameSize : number property (read-only)
Gets the Speex frame size

Decode (string buffer, var chunk_size=28)
Decodes the buffer. Chunk size is the Speex encoded chunk size. Returns the decoded Speex buffer as 16 bit PCM.

Note that Speex will remain active in the Concept Framework (will not be deprecated), but the use of Opus is recommended. Opus is a general-use codec, suitable for both speech and music, offering a higher compression than Speex while having a better quality.

Opus has a similar interface with Speex. It has the OpusEncoder and OpusDecoder, both defined in Opus.con

OpusEncoder:

Bitrate : number property
Sets the maximum bitrate to be used, in bits (default is 9000). It is similar to Speex' *Quality* parameter, but instead of a pre-defined value, the actual bit rate can be set.

Signal : number property
Sets the signal type. Default is OPUS_AUTO, but may be changed to OPUS_SIGNAL_VOICE or OPUS_SIGNAL_MUSIC.

Bandwidth : number property
Sets the voice bandwidth type (not the network). Default is OPUS_AUTO, but may be changed to OPUS_BANDWIDTH_NARROWBAND, OPUS_BANDWIDTH_MEDIUMBAND, OPUS_BANDWIDTH_WIDEBAND, OPUS_BANDWIDTH_SUPERWIDEBAND or OPUS_BANDWIDTH_FULLBAND.

OpusEncoder (samplerate=16000, channels=1, type=OPUS_APPLICATION_VOIP)
The constructor for OpusEncoder. *samplerate* is in hz, *channels* is the number of channels (1 for mono) and *type* is the application type. May also be: OPUS_APPLICATION_AUDIO, OPUS_APPLICATION_RESTRICTED_LOWDELAY.

Encode (string buffer, var out_size=null)
Encodes the buffer. *out_size* will be set to the output encoded size in bytes. Returns the encoded Opus buffer.

OpusDecoder:

SampleRate : number property (read-only)
Gets the sample rate in Hz

OpusDecoder (samplerate=16000, channels=1)
The constructor for OpusEncoder. *samplerate* is in hz and *channels* is the number of channels (1 for mono).

Decode (string buffer)
Decodes the given Opus encoded buffer. Returns the decoded buffer as 16 bit PCM data.

Note that both Opus and Speex import libraries implement g711 codecs for compatibility. See the Concept Framework documentation for standard.lib.opus for more information.

OpusEncodeDecodeExample.con

```
import standard.arch.opus
import standard.C.io

include Opus.con

class Main {
    Main() {
        var enc = new OpusEncoder(8000);
        var dec = new OpusDecoder(8000);

        var s=ReadFile("sample.raw");
        var len = length s;
        var opus_data = "";
        var pcm_decoded_data = "";

        for (var i=0;i<len;i+=320) {
            // get 160 samples (16 bits: 160 bytes * 2)
            var d = SubStr(s, i, 320);
            if (length d == 320) {
                // encode the data
                var out = enc.Encode(d);
                // decode the data
                var out2 = dec.Decode(out);
                opus_data += out;
                pcm_decoded_data += out2;
            }
        }
        WriteFile(opus_data, "out.opus");
        WriteFile(pcm_decoded_data, "out.raw");
    }
}
```

Note that for this example you need a file named *sample.raw* containing raw 16 bit, mono PCM data. For a file about 200kB, you should get an *out.opus* file of about 14kB.

The data from the Speex and Opus encoders are raw, as output by the encoders. It has no additional package or meta-data information associated. This task is up to the programmer.

13.2 Telephony and SIP integration

The Session Initiation Protocol (SIP) is a signaling communications protocol, widely used for controlling multimedia communication sessions such as voice and video calls over Internet Protocol (IP) networks.

The protocol defines the messages that are sent between peers which govern establishment, termination and other essential elements of a call. SIP can be used for creating, modifying and terminating sessions consisting of one or several media streams. SIP can be use for two-party (unicast) or multiparty (multicast) sessions. Other SIP applications include video conferencing, streaming multimedia distribution, instant messaging, presence information, file transfer, fax over IP and online games.[2].

For all SIP-based applications, the multi-threaded version of the core must be used (MT Core). The use of the non-threaded core, may cause crashes (refer to the Multi-threading section).

The SIP client class is *OPALSIP*, defined in OPALSIP.con. The OPALSIP interface is based on Open Phone Abstraction Library (OPAL), a C++ multi-platform, multi-protocol library for Fax, Video & Voice over IP and

[2] Source: http://en.wikipedia.org/wiki/Session_Initiation_Protocol on January 23, 2014

other networks[3].

Note that OPALSIP will create additional backgorund threads.

OPALSIP properties:

Quality number property
Sets the Speex codec quality. This is used in relation with the Concept RAudioStream object, not with SIP layer itself.

Bitrate number property
Sets the Opus codec maximum bitrate. This is used in relation with the Concept RAudioStream object, not with SIP layer itself.

Samplerate number property
Sets the sample rate (default 8000)

ConceptCodec number property
Sets the codec. May be CODEC_SPEEX(default) or CODEC_OPUS. This is used in relation with the Concept RAudioStream object, not with SIP layer itself.

LocalRTPPort number property (default 10002)
Sets the UDP port to be used for RTP streams

PortLimit number property (default 10)
Sets the UDP port interval. For example, if *LocalRTPPort* cannot be bound, the system will try to bind LocalRTPPort+1, +2 ... +10 before reporting an error.

MaxBuffer number property (default 500)
Sets the maximum buffer in milliseconds. When the buffer is over that value, it will be automatically cut to reduce jitter.

FrameSize number property (default 20)
Sets the audio frame size in milliseconds.

STUNServer string property
Sets a STUN server address. STUN (Session Traversal Utilities for NAT) is a standardized set of methods and a network protocol to

[3] http://www.opalvoip.org/

allow an end host to discover its public IP address if it is located behind a NAT[4].

Username string property
Sets the SIP username

Password string property
Sets the SIP password

LocalIP string property
Sets the local IP (tells the SIP server to send the messages to the given IP)

ProxyHost string property
Sets the host of the SIP server.

Codec number property (default RTP_PCMA)
Sets the RTP codec. May also be: RTP_PCMU, RTP_G721, RTP_G726, RTP_GSM, RTP_G722, RTP_G728 and other values. Check OPALSIP documentation for a complete list. Note that the code will be negotiated with the host.

Answered boolean property
Is set to true if the current call (invite request) was answered.

Incoming boolean property
Is set to true if the current call is incoming.

OPALSIP events:

OnDTMF (string character)
Called when a DTMF character (0123456789*#) is received.

OnEndCall (string target, number reason)
Called when a call ends. *reason* is the call end code. See SIP specification for a list with all codes.

OnCall (string target)
Called when an incoming call is received. *target* describes the calling user.

[4] http://en.wikipedia.org/wiki/STUN on January 23, 2014

OnVoice (string buffer, number payload_size)
Called when a voice packet was received. It is encoded using *ConceptCodec*. The buffer is the encoded data (Speex or Opus), and *payload_size* is the data size.

OnRinging (string target)
Called when an outgoing call was successfully initiated and the other party reported “Ringing”. *target* describes the other party.

OnRegister (number dummy)
Called when successfully registered with the SIP proxy.

OnAnswer (string target)
Called when the an outgoing call is answered. *target* describes the other party. *target* describes the other party.

OnConnect (caller)
Called when a the SIP proxy successfuly connects with the remote party.

OnAdjustBuffer ()
Called when the internal buffer is too big and must be reset. If this event returns true, the buffer will be reset to $frac13$ of its actual size.

OPALSIP methods:

ResetBuffer ()
Resets the internal buffer

Write (string buffer, number chunk_size)
Writes Speex or Opus-encoded data (depending on *ConceptCodec* value). Internally, the buffer is decoded and converted into the negotiated SIP codec.

Reject ()
Rejects or hangs up the current call

RejectCall ()
Rejects the current incoming call

Answer ()
Answers the current incoming call

Transfer (string user)
: Transfers the current call to another *user* and/or host.

Forward (string user)
: Forwards the current unanswered incoming call to another *user* and/or host.

SendTone (string character)
: Sends the given DTMF single tone (0123456789*#)

Hold ()
: Puts the current call on hold.

Hangup ()
: Hangups the current answered call

Register (number timeout=1000)
: Register to the SIP proxy using the *Username* and *Password* properties. Timeouts is the register retry timeout in milliseconds (not the actual register timeout).

Unregister ()
: Unregisters from the current SIP proxy

Call (string user)
: Initiates a call to the given *user* (with optional host).

array GetCodecs ()
: Gets a list of all SIP supported codecs and returns it as an array of strings

SetCodecs (array order)
: Sets the list of codecs (an array of strings). The codec on the first position (order[0]) has the maximum priority.

The simplest SIP client is shown in the following example:

```
include OPALSIP.con

class Main {
   var sip;

   OnEndCall(call, reason) {
      echo "End call: $reason\n";
```

```
    }

    OnRinging(to) {
        echo "Ringing $to\n";
    }

    OnCall(to) {
        echo "Call: $to\n";
    }

    Main() {
        sip=new OPALSIP();
        sip.Username = "yoursipusername";
        sip.Password = "passowrd";
        sip.ProxyHost = "sip.somehost.net";
        sip.Register();
        sip.OnEndCall = OnEndCall;
        sip.OnRinging = OnRinging;
        sip.OnCall = OnCall;
        // optinally, set the preferred codecs for the SIP connection
        sip.SetCodecs(["G.711-ALaw-64k", "G.711-uLaw-64k"]);
        echo sip.GetCodecs();
        sip.Register();
        Sleep(1000);
        echo "Call:";
        // if working with a Soft Switch
        echo sip.Call("123456789");
        // or
        // echo sip.Call("username@host.org");
        // wait 2.5 seconds, then end the call
        Sleep(2500);
        echo "Reject:";
        echo sip.Reject();
    }
}
```

Replace *yoursipusername*, *password* and *sip.somehost.net* with your username, password and SIP proxy host.

Note that the *OPALSIP* class can work directly with *RAudioStream* client class. This means that the audio stream is automatically converted to Speex or Opus, and the forwarded to the *RAudioStream* object. This is done via the *OPALSIP.OnVoice* event and *OPALSIP.Write* method.

The *RAudioStream* class, defined in OCV/RAudioStream.con provides access to the remote client microphone and speakers.

RAudioStream members:

SampleRate number property
Sets the sample rate, in Hz (default 44100).

Bitrate number property
Sets the bitrate to use with Opus codec (default is 9000)

Channels number property
Sets the channel number. Note that Speex will only work with *Channels* set to 1.

Compression number property
Sets the codec. Set it to 0 for plain PCM 16 bit data (not recommended), USE_SPEEX or USE_OPUS (recommended).

MaxBuffer number property (default 0)
Sets the maximum buffer count without resetting the internal buffer to avoid jitter. If set to 0, the property will be ignored

FrameSize number property
Sets the audio frame size in milliseconds (default 20).

DRM boolean property
If set to true, both the playback and recorded data will be encrypted using the key returned by DRMKey(). The encryption is AES-128 CBC. Note that DRM encrypting may not be supported on all platforms.

event OnBuffer (RAudioObject Sender, string buffer)
This event is called for every received audio buffer, encoded accordingly to *Compression* property, after a successfully call to *Record*(). If DRM is set to true, the buffer is also encrypted using AES-128 CBC and *DRMKey*() as a key.

Record (number device_id=-1)
Starts recording on the client using the given device_id. If set to -1, it will use the first available recording device.

Play (number device_id=-1)
Starts playback on the client using the given device_id. If set to -1, it will use the first available playback device. AddSmallBuffer

AddSmallBuffer (string buffer)
Adds a buffer for playback, encoded accordingly to *Compression* property. If DRM is set to true, the buffer must be encrypted using AES-128 CBC and *DRMKey*() as a key. It optimizes network traffic for buffer smaller than 0x9FFF. For other bigger than that, the call will fail.

AddBuffer (string buffer)
Non-optimized version of AddSmallBuffer. The use of AddSmallBuffer is encouraged.

AddBigBuffer (string buffer)
Similar to AddSmallBuffer, but allows multiple buffers. The use of AddSmallBuffer is encouraged.

Stop ()
Stops the current playback/recording

Note that you cannot call *Play* and *Record* simultaneous on the same object.

If you want to simply play a sound on the remote host, you may also use *RemoteAudio*, define in RemoteAudio.con.

```
include RemoteAudio.con
[..]
RemoteAudio::Go("res/hello.mp3");
[..]
```

The hello.mp3 must exist and be located in the concept application directory, in a subdirectory called “res”.

Both RemoteAudio and RAudioStream can only be used in concept:// applications (Concept Client-based).

```
include RemoteAudio.con
include OPALSIP.con
[..]
```

```
MainForm(owner) {
    super(owner);
    [..]
    this.Codec = USE_SPEEX;
    var codecs=ToLower(CApplication.Query("Codecs", false));
    // check if Opus codec is supported
    // note the ToLower modifier. Codec names sjould be
    // case insensitive

    if (Pos(codecs, "opus")>0)
        Codec=USE_OPUS;

    raudio_out = new RAudioStreamer(this);
    raudio_out.SampleRate = 8000;
    raudio_out.Channels = 1;
    raudio_out.MaxBuffers=10;
    raudio_out.Compression=Codec;

    raudio_in = new RAudioStreamer(this);
    raudio_in.OnBuffer = this.onConceptClientBuffer;
    raudio_in.Channels = 1;
    raudio_in.Quality = 9;
    raudio_in.Compression = Codec;
    raudio_in.SampleRate = raudio.SampleRate;

    SIP = new OPALSIP();
    SIP.OnVoice = this.onSIPBuffer;
    [..]
}

[..]

onAnswerButtonPressed(Sender, EvnetData) {
    // assume we have a button to press to answer a call
    raudio_in.Record();
    raudio_out.Play();
}

onHangupButtonPressed(Sender, EvnetData) {
    // assume we have a button to press to answer a call
    SIP.Reject();
    raudio_in.Stop();
    raudio_out.Stop();
}
```

```
    onSIPBuffer(string buffer, number chunk\_size) {
        // note that this function will NOT be called in the main
            loop
        // the call will originate from an OPALSIP thread
        SendAPMessage(GetAPID(), 101, buffer);
    }

    OnInterAppMessage(Sender, MSGID, data) {
        if (MSGID == 101) {
            // buffer received from another thread
            if (length data<0x9FFF)
                raudio.AddSmallBuffer(data);
            else
                raudio.AddBuffer(data);
        }
    }

    onConceptClientBuffer(Sender, buffer) {
        if (this.Codec == USE_SPEEX) {
            // ignore fist byte (the buffers count)
            buffer++;
            // speex buffers received from the client
            // have chunk_size on the second byte
            var chunk_size = ord(buffer[0]);
            // make buffer start from the next character
            buffer++;
            SIP.Write(buffer, chunk_size);
        } else {
            // its simpler with Opus
            SIP.Write(buffer, length buffer);
        }
    }
[..]
```

The example only illustrates the concepts behind *OPALSIP* and *RAudioStream*. For a complete VoIP (without SIP) example, check the *OpenConference* in the Concept distribution's Samples directory.

Note that *RAudioStream* is highly optimized in mobile version of Concept Client (Android and iOS), enabling you to create VoIP applications that use only the concept:// protocol, both for voice and UI.

Concept also supports basic DTMF recognition by using the

DTMFDetector class.

DetectDTMF.con

```
include RAudioStream.con
include DTMFDetector.con
include RForm.con

define APPLICATION_FORM_TYPE MainForm

class MainForm extends RForm {
    var audio;
    var decoder;

    public function MainForm(Owner) {
        super(Owner);

        audio = new RAudioStreamer(this);
        audio.Compression = false;
        audio.Channels = 1;
        audio.SampleRate = 8000;

        audio.OnBuffer = function(Sender, EventData) {
            decoder.AddBuffer(EventData);
            var b = decoder.Buttons;
            if ((b) && (b != label1.Caption))
                label1.Caption = b;
        };
        decoder=new DTMFDetector();
        audio.Record();
    }
}
```

13.3 Digital rights management

Concept Framework provides a simple DRM (Digital Rights Management) engine to provide secure audio data exchange between CAS and Concept Client. This is simply done by setting the *RAudioStream.DRM* property to true.

The actual rights settings are up to the programmer. The DRM system ensures a secured packet exchange with a secure symmetric key, negotiated between the Concept Client and CAS. The used algorithm is AES 128 CBC, and the key is returned by the DRMKey().

The DRM engine can be also used for secured concept:// Voice over IP applications.

```
include OPALSIP.con
import standard.lib.cripto
[..]
    InitDRM(owner) {
        var DKey = DRMKey();
        if (DKey) {
            if (context_in)
                AESRelease(context_in);
            if (context_out)
                AESRelease(context_out);

            context_in = AESDecryptInit(DKey);
            context_out = AESEncryptInit(DKey);
        }
    }

    onSIPBuffer(string buffer, number chunk\_size) {
        var buffer=AESEncrypt(context_out, buffer, BLOCKMODE_CBC,
            true);
        [..]
    }

    onConceptClientBuffer(Sender, buffer) {
        buffer = AESDecrypt(context_in, buffer, BLOCKMODE_CBC, true);
        [..]
    }
[..]
```

Apart calling AESEncrypt and AESDecrypt, the rest of the API calls remain exactly the same as the non-DRM version. If DRM is set to 2 instead of true (1), then the BLOCKMODE_ECB shall be used.

13.4 Image and video processing

Concept has a few image processing import libraries. The most used is *win32.graph.freeimage*, the most advanced is *standard.graph.imagemagick* and the most useful is *standard.graph.svg*.

The freeimage import library is based on the FreeImage project (open source). The "win32" prefix exists for backwards-compatibility only, the library being portable and working on most operating systems.

It can modify, resize, rotate or convert an image from and to usual formats. It has lots of APIs. For a complete list, check the Concept Framework Documentation.

FreeImageResizeExample.con

```
import win32.graph.freeimage
import standard.C.io
import standard.lib.str

class Main {
    static GenerateThumb(string image_name, string thumb_name,
        number max_w=300, number max_h=200, string
        img_type=FIF_JPEG, number crop=true) {
        var type = FreeImage_GetFileType(image_name);
        image_type = ToLower(image_type);
        if (type < 0)
            return false;
        var hBitmap = FreeImage_Load(type, image_name, 0);
        if (!hBitmap)
            reuturn false;
        var true_width = FreeImage_GetWidth(hBitmap);
        var true_height = FreeImage_GetHeight(hBitmap);
        if ((true_width <= max_w) && (true_height <= max_h)) {
            if (!FreeImage_Save(img_type, hBitmap, thumb_name, 0)) {
                FreeImage_Unload(hBitmap);
                return false;
            }
            FreeImage_Unload(hBitmap);
            return true;
        }
        var aspect_ratio = 1;
        if ((!true_width) || (!true_height)) {
            FreeImage_Unload(hBitmap);
            return false;
        }
        var coef_w = max_w / true_width;
        var coef_h = max_h / true_height;
        if ((crop) && (true_height > max_h) && (true_width > max_w)) {
            var p_a_ratio = true_width / true_height;
            var t_a_ratio = max_w / max_h;
            if (p_a_ratio != t_a_ratio) {
                var new_height = true_height;
                var new_width = true_width;
                if (coef_h>coef_w)
                    new_width = true_height * t_a_ratio;
                else
                    new_height = true_width / t_a_ratio;
                if (new_width > true_width)
                    new_width = true_width;
                if (new_height > true_height)
                    new_height = true_height;
                var left = (true_width-new_width) / 2;
                var right = true_width-left;
                var top = (true_height-new_height) / 2;
                var bottom = true_height-top;

                var hBitmap3 = FreeImage_Copy(hBitmap, left, top, right, bottom);
                if (hBitmap3) {
                    FreeImage_Unload(hBitmap);
                    hBitmap = hBitmap3;
                    hBitmap3 = null;
                    true_width = new_width;
                    true_height = new_height;
                    coef_w = max_w/true_width;
                    coef_h = max_h/true_height;
                }
            }
        }
        if (coef_w<coef_h)
            aspect_ratio *= coef_w;
        else
            aspect_ratio *= coef_h;
        if (aspect_ratio > 1)
            aspect_ratio = 1;

        var hBitmap2 = FreeImage_Rescale(hBitmap, true_width *
            aspect_ratio, true_height * aspect_ratio, 0);

        FreeImage_Unload(hBitmap);
        if (hBitmap2) {
            if (!FreeImage_Save(img_type, hBitmap2, thumb_name, 0)) {
                FreeImage_Unload(hBitmap2);
                return false;
            }
        }
        FreeImage_Unload(hBitmap2);
        return true;
    }

    Main() {
        GenerateThumb("test.jpg", "thumb.png", 100, 100, FIF_PNG);
    }
}
```

The previous example resize a JPG to 100x100 pixels, converts it to PNG format and performs a crop operation. In practice avoid writing such functions performing multiple operations, because debugging can become difficult.

The same task can be done by using the more powerful *standard.graph.imagemagick* import library. In addition, it provides image creation APIs, not just manipulation.

MagickExample.con

```
import standard.graph.imagemagick
import standard.C.io

class Main {
    function Main() {
        var w=NewMagickWand();
        var img=MagickReadImage(w, "test.png");
        if (!img) {
            echo MagickGetException(w, var sev);
            return -1;
        }
        // We could resize the image:
        // MagickResizeImage(w, 32, 32, 0, 1);
        var draw=NewDrawingWand();
        // you could get a list with all the available font with
        // var fonts_arr = MagickQueryFonts("*");
        DrawSetFont(draw, "test.ttf");
        DrawSetFontSize(draw, 50);
        MagickAnnotateImage(w, draw, 50, 250, -15, "A cool
            butterfly");
        MagickWriteImage(w, "test2.png");
        WriteFile(MagickGetImageBlob(w),"test3.png");
        DestroyMagickWand(w);
    }
}
```

Assuming that we have *test.ttf* and *test.png* in the application directory, the output may look like figure 13.2.

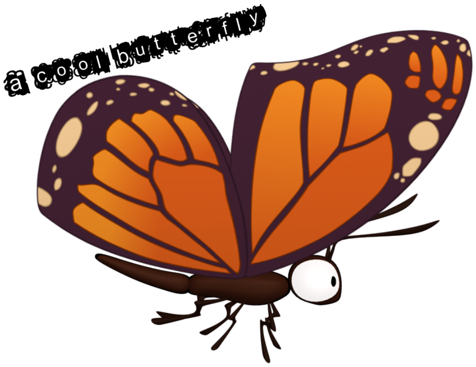


Figure 13.2:
MagickExample.con

Concept has support for SVG files. Scalable Vector Graphics (SVG) is an XML-based vector image format. *The standard.graph.svg* implements just one static functions:

```
string SVG(string svg_buffer, type="png", dpi=-1, var error=null);
```

The input is the SVG file content (XML), the output is the image buffer, in the format specified by *type*. If given, the error will contain the parsing error.

Two concept classes generate SVG output: *LineChart*, defined in LineChart.con and *PieChart*, defined in PieChart.con.

RandomLineChart.con

```
include LineChart.con

import standard.C.io
import standard.graph.svg

class Main {
    function Main() {
        var data=new [];
        // first line: the series names
        data[0]=["Series 1", "Series 2"];
        // row format: point name, series1 value, ..., seriesN value
        // int32 rounds a double to a value that could be contained
        // in an 32 bit signed integer
        for (var i=0;i<150;i++)
            data[length data]=["val"+int32(i/10), rand(), rand()];

        var svg=LineChart::Do(data, 1000, 300, 1, 2, "white");
        // save the SVG output
        WriteFile(svg, "out.svg");
        // save output as PNG
        WriteFile(SVG(svg), "out.png");
    }
}
```

The output will be similar with 13.3.

The standard.graph.svgt implements Tiny SVG, a lightweight version of SVG, also implementing just one static function:

```
string SVGT(string svgt_buffer, number scale=1.0);
```

However, you should avoid using SVGT in favor of SVG. SVGT should be used only on embedded devices, where the SVG overhead is considered to

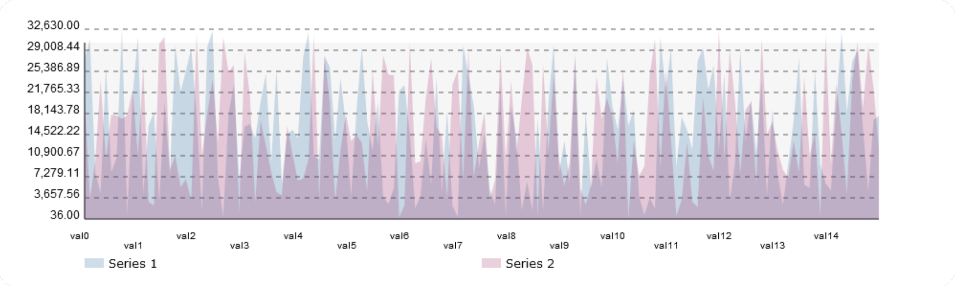


Figure 13.3:
RandomLineChart.con output

be heavy.

Concept has hundreds of image manipulating APIs. Check the Concept Framework documentation for a complete list.

For video processing, a wrapped class for the *ffmpeg* command line utility called *FFMpeg*, defined in FFMpeg.con, provides access to video files.

It can extract frames, get video information and perform conversion between different formats. This can be used for video normalization (keeping videos in the same format, regardless of the uploaded format). Also, video frames can be easily extracted by sing the *GetPreview* function. This is useful for generating video preview image.

```
include FFMpeg.con

class Main {
    function Main() {
        var ff=new FFMpeg();
        try {
            ff.InputMovie="video.avi";
            echo "Duration  : " + ff.Duration + " seconds\n";
            echo "Resolution : " + ff.VideoInfo[VIDEO_SIZE] + "\n";
            echo "Video codec: " + ff.VideoInfo[VIDEO_CODEC] + "\n";
            echo "Audio codec: " + ff.AudioInfo[AUDIO_CODEC] + "\n";

            echo "\n\nStarting generating best previews ...\n";
            var idcode="best_preview";
            var idcode_png=idcode+".png";
            var idcode_mpg=idcode+".mpeg";

            // generate a preview. Identify 5 previws, and
            // choose the one with most colors (avoiding black
                frames)
            ff.GetPreview(idcode_png, PREVIEW_PNG, "320x240", 5);

            echo "Converting to MPEG";
            ff.Convert(idcode_mpg);

        } catch (var exc) {
            echo exc;
        }
    }
}
```

The *FFMpeg* class provides only basic video information, frame extraction and video conversion. In practice, the use of *ffmpeg* utility with the *system* static function (defined in *standard.C.io* import library) is recommended.

standard.lib.captcha has the Captcha(var text) function, returning the contents of a gif file and setting the text to a random string, for generating CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart).

```
import standard.lib.captcha
import standard.C.io

class Main {
    Main() {
        WriteFile(Captcha(var data), "captcha.gif");
    }
}
```

The resulting file is shown in figure 13.4.

PDF files can be easily read with the *standard.lib.poppler* import library. You can extract text, images or convert pages to images.

The following static functions are provided:

handle PDFLoadBuffer (string buffer, string password="", var error=null)

Figure 13.4:
Generated file (captcha.gif)

Loads a PDF from the given string *buffer*. If *password* is set, the given password is used for decrypting the document. *error* (if given) will be set to the error message, if any. On succes it returns a document handle. If it fails, it returns null.

handle PDFLoad (string filename, string password="", var error=null)
Loads a PDF from the given file. If *password* is set, the given password is used for decrypting the document. *error* (if given) will be set to the error message, if any. On succes it returns a document handle. If it fails, it returns null.

PDFClose (handle)
Closes the PDF.

number PDFPageCount (handle)
Returns the number of pages in the loaded document.

string PDFPageText (handle, number pageindex)
Returns the text, as a Concept string, contained by the PDF on the given page index. Page 0 is the first page.

array PDFFindText (handle, number pageindex, string what)
Searches for *what* on the given page. Returns an array containing key-value arrays as elements, describing the position of the text in the PDF. The keys are: *x1, y1, x2* and *y2*. If the function fails, it returns null.

string PDFImageBuffer (handle, number pageindex, type="png", zoom=1.0, var error=null)

number PDFImage (handle, number pageindex, string filename, type="png", zoom=1.0, var error=null)
Returns true if succeeded.

array PDFAttachments (handle)

Returns the attached PDF objects, as an array of arrays. Each array element has the following keys: *name, description, size, ctime, mtime* and *content.*

It is fairly easy to read a PDF document:

PDFReaderExample.con

```
import standard.lib.poppler
import standard.C.io

class Main {
    function Main() {
        var pdf = PDFLoadBuffer(ReadFile("test.pdf"), "", var err);
        if (pdf) {
            var pages = PDFPageCount(pdf);
            echo "Document has $pages pages\n";
            for (var i = 0; i < pages; i++) {
                echo "Page ${i+1}:\n";
                echo "=============================\n";
                // show page text
                echo PDFPageText(pdf, i);
                // save page as image
                PDFPageImage(pdf, i, "page_$i.png");
                echo "=============================\n";
                // search some text
                var arr = PDFFindText(pdf, i, "sample");
                if (arr)
                    echo "It contains the searched word!\n";
            }
            PDFClose(pdf);
        }
    }
}
```

The previous example will load a PDF file from a file and will convert it to a sequence of images. It will also print the contents of the pages and search for the word “sample” on every page.

13.5 Face detection

The *standard.lib.face* provides face detection routines based on OpenCV. It also provides access to CAS capturing sources, for example web cams, (the one on the server-side), for capturing images from various sources.

handle LoadObject (string filename)
Loads a Haar Feature-based Cascade Classifier for Object Detection from filename. Returns a handle (not null) if succeeded.

CloseObject (number handle)
Closes the loaded object, and frees the memory asociated with it

array ObjectDetect (string inputname, harr_handle_or_filename[, number type=OBJECT_FILE, biggest_object=false, string format, number width, number height, iterations = 5])
Detects an ojbect. Returns an array containing the detected objects, if succeeded, or a number describing the error code if failed. Input name may be a image filename or a camera index (or an empty for any camera) if the type is set to OBJECT_CAMERA. If set to OBJECT_BUFFER, the *inputname* should be the actual buffer.

array FaceRecognize (string inputname, array training[, var confidence, type=OBJECT_FILE, string trainingdata_filename, var trainig_container)
Detects a face in the given input/type. Returns a negative number on error.

CloseRecognize (handle)
Closes the training data handle.

string ReadCamera (camera_index=0, format="png", delay=0)
Captures an image from the given camera index and returns the image buffer as a string in the given format.

handle OpenCamera (camera_index=0[, number width, number height])
Opens a camera. If *width* and *height* are provided, will open it at the given resolution

CloseCamera (handle)
Closes a previously opened camera.

string CameraFrame (handle, format="png")

Captures a frame from the given camera, and returns the captured frame as a buffer, in the given format.

```
import standard.lib.face
import standard.C.io
import standard.graph.imagemagick

class Main {
    function Main() {
        var data=ReadFile("test.jpg");
        // we could also capture an image from any camera
        // on the server
        // var data = ReadCamera();
        var arr=ObjectDetect(ReadFile("people.jpg"), "face.xml",
            OBJECT_BUFFER, 0);

        // the dection is over, now draw some rectangles
        var w = NewMagickWand();
        var img = MagickReadImage(w, "people.jpg");
        var draw = NewDrawingWand();
        var color = NewPixelWand();
        PixelSetColor(color,"#00ff00");
        DrawSetStrokeWidth(draw, 3);
        DrawSetStrokeColor(draw, color);
        var color2 = NewPixelWand();
        PixelSetColor(color2,"none");
        DrawSetFillColor(draw, color2);
        var len = length arr;
        for (var i=0; i<len; i++) {
            var obj = arr[i];
            if (obj) {
                var x = obj["x"];
                var y = obj["y"];
                DrawRectangle(draw, x, y, x + obj["w"], y + obj["h"]);
            }
        }
        MagickDrawImage(w, draw);
        MagickWriteImage(w, "detected.png");
    }
}
```

face.xml defined an object for face recognition. You will find it with a simple web search on face.xml. The output file (detected.png) is shown in figure 13.5.

You could also identify a face, by using:

```
var result = FaceRecognize(ReadFile("frame.png"), ["training1.png",
    "training2.png", "training3.png"], var confidence,
    OBJECT_BUFFER, "", var container);
// or
var result = FaceRecognize(ReadCamera(), ["training1.png",
    "training2.png", "training3.png"], var confidence, OBJECT_CAM,
    "", var container);
```

Result will be < 0 on error, false if not recognized or > 0 if input data matches the training data.

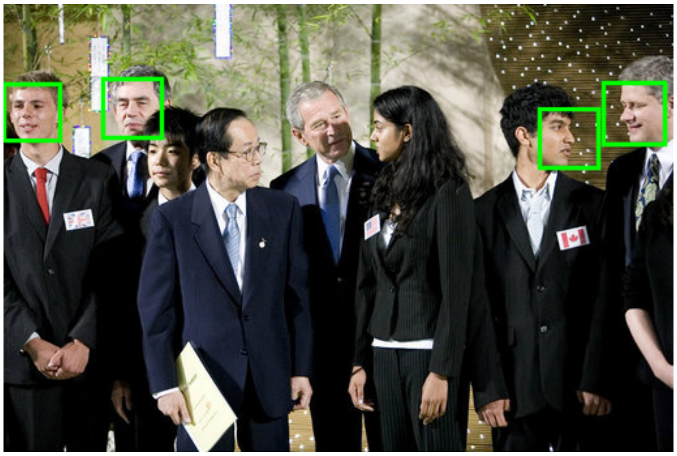


Figure 13.5:

For capturing images from the Concept Client, see the documentation for the ROCV class.

Note that *ObjectDetect* is not limited to faces. You may use it with any object definition.

13.6 H.264 video support

Concept supports the H.264 video codec. In the future, support for HEVC (H.265), VP8 and VP9 will be added. For now, the support is limited to H.264 due to the hardware support available in different devices. For example, Intel Haswell CPUs support hardware encoding and decoding. Also, various smart phones and tables, eg. recent iPhones (starting with fifth generation), and some high-end Android devices have hardware H.264 encoders. This minimizes the power usage while encoding or decoding H.264 frames.

On the server-side Concept H.264 support is based on Cisco's OpenH264 project. It was primarily chosen because the BSD license, that allows deployment in commercial applications.

The server-side codec library (standard.arch.h264) works only with YUV420p frames. YUV is a color space typically used as part of a color image pipeline. It encodes a color image or video taking human perception into account, allowing reduced bandwidth for chrominance components, thereby typically enabling transmission errors or compression artifacts to be more efficiently masked by the human perception than using a "direct" RGB-representation.[5].

standard.arch.h264 API list:

handle H264Encoder (array properties)
Creates a H.264 encoder

handle H264Decoder (array properties)
Creates a H.264 decoder

H264EncoderSet (handle encoder, number property, number value)
Sets a property to the *encoder*. *property* must be one of:
ENCODER_OPTION_DATAFORMAT,
ENCODER_OPTION_IDR_INTERVAL,
ENCODER_OPTION_SVC_ENCODE_PARAM_BASE,
ENCODER_OPTION_SVC_ENCODE_PARAM_EXT,
ENCODER_OPTION_FRAME_RATE,
ENCODER_OPTION_BITRATE,

[5] Source:https://en.wikipedia.org/wiki/YUV as shown on August 7, 2015

ENCODER_OPTION_MAX_BITRATE,
ENCODER_OPTION_INTER_SPATIAL_PRED,
ENCODER_OPTION_RC_MODE,
ENCODER_OPTION_RC_FRAME_SKIP,
ENCODER_PADDING_PADDING,
ENCODER_OPTION_PROFILE, ENCODER_OPTION_LEVEL,
ENCODER_OPTION_NUMBER_REF,
ENCODER_OPTION_DELIVERY_STATUS,
ENCODER_LTR_RECOVERY_REQUEST,
ENCODER_LTR_MARKING_FEEDBACK,
ENCODER_LTR_MARKING_PERIOD, ENCODER_OPTION_LTR,
ENCODER_OPTION_COMPLEXITY,
ENCODER_OPTION_ENABLE_SSEI,
ENCODER_OPTION_ENABLE_PREFIX_NAL_ADDING,
ENCODER_OPTION_ENABLE_SPS_PPS_ID_ADDITION,
ENCODER_OPTION_CURRENT_PATH,
ENCODER_OPTION_DUMP_FILE,
ENCODER_OPTION_TRACE_LEVEL,
ENCODER_OPTION_TRACE_CALLBACK_CONTEXT,
ENCODER_OPTION_GET_STATISTICS,
ENCODER_OPTION_STATISTICS_LOG_INTERVAL,
ENCODER_OPTION_IS_LOSSLESS_LINK,
ENCODER_OPTION_BITS_VARY_PERCENTAGE

H264EncoderGet (handle encoder, number property, out number value)
Gets an encoder property and sets the *value* parameter

H264DecoderSet (handle encoder, number property, number value)
Sets a property to the *decoder*. *property* must be one of:
DECODER_OPTION_DATAFORMAT,
DECODER_OPTION_END_OF_STREAM,
DECODER_OPTION_VCL_NAL,
DECODER_OPTION_TEMPORAL_ID,
DECODER_OPTION_FRAME_NUM,
DECODER_OPTION_IDR_PIC_ID,
DECODER_OPTION_LTR_MARKING_FLAG,
DECODER_OPTION_LTR_MARKED_FRAME_NUM,
DECODER_OPTION_ERROR_CON_IDC,
DECODER_OPTION_TRACE_LEVEL,
DECODER_OPTION_TRACE_CALLBACK_CONTEXT,

DECODER_OPTION_GET_STATISTICS

H264DecoderGet (handle encoder, number property, out number value)
Gets an decoder property and sets the *value* parameter.

H264EncoderDone (handle encoder)
Releases the memory used by the encoder. Not calling this function after encoding is finished will result in a memory leak.

H264DecoderDone (handle decoder)
Releases the memory used by the decoder. Not calling this function after decoding is finished will result in a memory leak.

number H264Encode (handle encoder, yuv420p_buffer, image_width, image_height, format=videoFormatI420, frame_timestamp = 0)
Encodes yuv420p_buffer. The buffer size must be image_width * image_height * 3 / 2 (or 12 bits per pixel). On success, it returns 0.

number H264Decoder (handle decoder, h264_buffer, out yuv420p_output, number_of_frames = 0, out number frame_width = 0, out number frame_height = 0, out number format = videoFormatI420)
Decodes a *h264_buffer* or NAL unit and sets *yuv420p_output*. If *number_of_frames* is non-zero, and the function call succeeds *yuv420p_output* will contain an array of yuv420p frames. If *number_of_frames* is zero, *yuv420p_output* will contain a buffer with a single frame. Optionally this function will return the frame width and height, and picture format. If it succeeds, it will return 0.

The simplest example is described bellow. Before running it, be sure to place a plain YUV420p video file in the *./in/outHD.yuv* directory and create a directory called *out*.

For converting a video to yuv format, ffmpeg may be used:

```
ffmpeg -i 10secondsHDVideo.avi -c:v rawvideo -pix_fmt yuv420p
    ./in/outHD.yuv
```

H264EncodeDecodeExample.con

```
import standard.arch.h264
include File.con
```

```

class Main {
    Main() {
        var width = 1920;
        var height = 1080;

        var enc = H264Encoder([ "iPicWidth" => width,
            "iPicHeight" => height,
            "fMaxFrameRate" => 25,
            "iTargetBitrate" => 10000000,
            "bEnableSSEI" => true,
            "iMultipleThreadIdc" => 4,
            "bSimulcastAVC" => true
        ]);
        var dec = H264Decoder(["sVideoProperty.eVideoBsType" =>
            VIDEO_BITSTREAM_AVC, "eOutputColorFormat" =>
            videoFormatI420]);
        var ref = "";
        var f = new File("rb");
        f.Name = "in/outHD.yuv";
        if (f.Open()) {
            for (var i = 0; i < 1000; i++) {
                f.Read(var buf, width * height * 3/2);
                if (!buf)
                    break;
                if (H264Encode(enc, buf, width, height, var out))
                    echo "Error encoding frame\n";

                ref+=out;
                if (out) {
                    var res = H264Decode(dec, out, var b2);
                    if (res)
                        echo "Error decoding frame\n";
                    WriteFile(b2, "out/frame$i.yuv");
                }
                echo "Frame: $i\n";
            }
            f.Close();
            WriteFile(ref, "out.h264");
        }
        H264EncoderDone(enc);
        H264DecoderDone(dec);
    }
}
```

This will encode and then decode a raw YUV420p video.

Note that Concept Client supports only H.264 AVC.

For capturing video from the Concept Client (mobile, and web browsers) the **RVideoStream** class, defined in *OCV/RVideoStream.con* may be used. A single **RVideoStream** object should be used both for recording and playing video content.

Note: This class works only with plain H.264 NAL units. This means that containers like mp4, avi or ts can't be used.

RVideoStream class reference:

RVideoStream (owner)
Creates a video stream object

Record ()
Start capturing H.264 frames from camera

Stop ()
Stops camera capturing

AddBuffer (buffer)
Adds a H.264 frame buffer to the camera playback queue.

Clear ()
Clears the playback queue, discarding any buffers that didn't play.

event OnBuffer (Sender, buffer)
Called when a new H.264 buffer is available from the camera

number property Bitrate
Sets the bitrate of the output buffer. Note that the actual bitrate may vary depending the the device's implementation

number property Quality
Sets the video quality, 10 being the highest (HD) and 4 the lowest quality usually fit for video conferences over 3G or congested networks.

number property FrameSize
Sets the frame duration in milliseconds. For example, for 25 frames per seconds, FrameSize will be 40ms.

number property Camera = CAMERA_BACK — CAMERA_FRONT
Selects the back facing or front facing camera.

number property Orientation = ORIENTATION_HORIZONTAL — ORIENTATION_VERTICAL
Sets the camera orientation.

number property MaxBuffers
Sets the maximum buffers to keep in the playback queue. If this value is 0, the number of buffers in unlimited.

VisibleRemoteObject property Preview
Selects the preview and playback surface to be used.

SetPreviewRect (x, y, width, height)
Sets the preview rectangle to be used when recording video. If not set, the preview will use the entire *Preview* surface.

The following example wil capture data from the front facing camera and output it back. It will use the highest resolution and 3 megabits bandwidth.

SimpleCameraMirror.con

```
include Application.con
include RForm.con
include OCV/RVideoStream.con

class MyForm extends RForm {
    var camera;
    MyForm(parent) {
        super(parent);
        camera = new RVideoStream(this);
        // select front camera
        camera.Camera = CAMERA_FRONT;
        // select best quality
        camera.Quality = 10;
        camera.Orientation = ORIENTATION_VERTICAL;
        // set bitrate
        camera.Bitrate = 3000000;
        // set the preview rectangle size (upper left)
        camera.SetPreviewRect(10,20,48,64);
        // use this form as the preview surface
        camera.Preview = this;
        // max playback queue size is 20 frames
        camera.MaxBuffers = 20;
        camera.OnBuffer = function(Sender, EventData) {
            // play the capture data
            camera.AddBuffer(EventData);
        };

        this.Maximized = true;
        camera.Record();
    }
}

class Main {
    function Main() {
        try {
            var Application=new CApplication(new MyForm(NULL));
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            echo Exception;
        }
    }
}
```

Chapter 14

Performance

When writing code, in any language, not just Concept, is important to write it cleanly and as optimal as possible. A program will spend most of its time in loops. That is why the inner loop code must be written carefully. In critical sections of the loop, or even the loop itself, it is important to keep in mind that the code will be compiled by the JIT. The JIT-compiled code will exit on function calls, a member reference or a string modification. When the JIT will exit, the Concept Core interpreter will process the byte code, at only a fraction of the speed. Note that every called function, when called for the first time, is entirely interpreted. The Concept Core Interpreter is a fast interpreter, but the JIT code is faster (as any binary code). A function is compiled on the second call and executed as machine code (native code), for optimizing the memory usage. If all the functions were compiled, it will use with about 20% more memory, and usually Concept applications run on a server, where the memory is important. Less memory used, means more users served by the server.

When analyzing the operating systems, the Concept Core and the CAS itself, has a slight advantage on non-Windows operating systems, but this may come from better networking and disk caching implementations on BSD and Linux.

In most applications the bottleneck will be the network speed, database access or disk I/O. Concept Core itself reasonably fast, and when running on JIT is comparable with code generated with C/C++ compilers.

14.1 Optimal loops

Most programs will spend most of their running time in loops. A loop, if not written properly, may be a bottleneck for an application.

The first step is to optimize the exit condition. For example, the “bad” loop:

```
for (var i = 0; i < length v; i++)
    v[i] = 2;
```

The length operator in Concept is highly optimized, but it still generates an extra CAL instruction, resulting into a less than optimal loop. Also, for the given example, the JIT optimizing algorithms will not be able to detect that it is an array initialization. The optimized loop should look like this:

```
var len = length v;
for (var i = 0; i < len; i++)
    v[i] = 2;
```

Note that *for*, *while* and *do..while* in CAL byte code produce almost identical byte code, so its up to the programmer preferences which one to use.

Member access use additional instructions, making them a little slower than local variables, so when using multiple times the same variable member, you may want to store its value it in a local variable.

The “bad” example (adding to an already initialized member the sum of 1 to 100):

```
var i = 0;
while (++i <= 100)
    this.SomeValue += i;
```

For every loop, the member will be read. This will also produce a break in the JIT code for every read or write of the *SomeValue* member. The optimal loop would be:

```
var i = 0;
var MemberValue = this.SomeValue;

while (++i <= 100)
    MemberValue += i;

this.SomeValue = MemberValue;
```

Note that when running the MT Core (multi-threaded), the *this.SomeValue* value may be changed from another thread. For multi-threading, the previous example may produce inconsistence, if *this.SomeValue* would be modified from another thread. For this, you have semaphores (refer to section 12.2).

Similar with member access, array element access require extra instruction in the compiled code. Also, arrays with less than 8192 elements are highly optimized for access. For array bigger than 8192, is a little faster to access elements in sequential order, rather than random order. Arrays implement a mini-caching system, that cache the last element, resulting in an almost zero overhead when performing operations like *arr[n] = arr[n] * 2 + 1*. However, you may want to store its value in a local variable, to optimize high time-consuming loops.

For example, the "bad" loop:

```
var i = 0;
while (++i < 100)
    arr[i] = arr[i] * arr[i] * arr[i];
```

The loop will generate only one extra instruction in the CAL, but if your loop is run million times per second, the following loop, may reduce the execution time with about 10%.

```
var i = 0;
while (++i < 100) {
    var e = arr[i];
    arr[i] = e * e * e;
}
```

We could optimize the loop even further, if we reduce the number of operations by one:

```
var i = 0;
while (++i < 100) {
    var e = arr[i];
    arr[i] *= e * e;
}
```

Note that instead of making two multiply operations, we make one and an assignment with multiplication (one instruction). This kind of optimizations, gives you a slight execution time improvement (about 5 to 10 percent).

Static Concept functions require an extra step in their execution than standard functions. This is because a virtual object is created to hold the function. So assuming that we have:

```
class SomeClass {
    static foo() {
        do_something();
    }
}
```

SomeClass::foo() will execute a little slower than if we'd execute *this.foo*(). The difference is barely noticeable, but if *foo* is called millions of times, this could add up.

When performing operations between different data types, avoid letting the Concept Core evaluating constant strings in loops, because it will be re-evaluated for every loop. For example, the "bad" loop:

```
var i = 0;
var sum = 0;
while (++i <= 100)
    sum += "1";
```

sum, being a number, will add the "1" string as a number, not as a string. If sum would be initialized to "" instead of 0, then will create an array containing 100 of "1". For every iteration, "1" will be evaluated to its corresponding number value, 1. The optimal loop may look like this:

```
var i = 0;
var sum = 0;
var constant = value "1";
// constant will be 1, the number
while (++i <= 100)
    sum += constant;
```

Concept properties also require an extra step when evaluating (calling the get member). When using the same property, multiple times, you may want to store its value in a local variable, exactly like in the member access example. Also, avoid setting it multiple times in a loop. This will cause multiple calls to set. When dealing with Concept UI class properties, is may generate a client property request, resulting in network traffic, being extremely important to store the property variable in a local network, to avoid an unnecessary network traffic.

The “bad” UI property example:

```
include REdit.con
[..]
    var dummy = edit.Text + edit.Text;
[..]
```

In the above example, each call to *edit.Text* will generate a network message and will wait for the response. This will last from one millisecond, to a few hundreds milliseconds, depending on the network connection quality. Reading the *Text* property twice, will generate two request and two wait operations. The optimal and correct way to do this is:

```
include REdit.con
[..]
    var text = edit.Text;
    var dummy = text + text;
[..]
```

This will generate only one network message and only wait operation. The UI property set operation does not generate a wait operation, being significantly faster than get. This applies only for UI objects (The one prefixed by “R” for example *REdit* or *RImage*).

When setting multiple UI class properties, the networks messages can be grouped in a jumbo packet, containing more than one message, by enclosing the property set *CApplication::BeginNonCritical*() and *CApplication::EndNonCritical*(). Note that the call to *EndNonCritical* is very important. Not calling it will cause a very high latency in set operations. These calls are effective when setting more than 4 properties in a row. A read of an UI property will fragment the jumbo packet, assuming that it cannot read a property until all the cached write operations are not finished.

The “bad” example:

```
include REdit.con
include RLabel.con
[..]
    label.Caption = "Name:";
    edit1.Text = "Your name";
    label2.Caption = "Occupation:";
    edit2.Text = "Your occupation";
[..]
```

Each of the set operations will cause small network packages to be sent to the client. You can send all the Concept messages in one big network package by using:

```
include REdit.con
include RLabel.con
[..]
    CApplication::BeginNonCritical();
    label1.Caption = "Name:";
    edit1.Text = "Your name";
    label2.Caption = "Occupation:";
    edit2.Text = "Your occupation";
    CApplication::EndNonCritical();
[..]
```

Note that *BeginNonCritical* and *EndNonCritical* will generate two more Concept messages. A code block must have at least 4 property set operations in order for this to be network efficient. This is useful for big data forms, with tens of fields, resulting in all the data being set in one operation. It is important to avoid calling *BeginNonCritical* twice, without calling *EndNonCritical.* This may cause the message jumbo packet to contain only a few messages and standard packets to be used.

Avoid using empty loops, for example:

```
while (this.SomeFlag < 100);
```

Assuming that you're in a multi-threaded environment (MT Core), and *SomeFlag* will be modified by another thread. For this, you can use semaphores (refer to section 12.2). Alternatively, you may introduce a Sleep in the loop, to minimize the CPU usage for the current thread.

```
while (this.SomeFlag < 100)
    Sleep(50);
```

This will cause the thread to waste 50 milliseconds, giving other threads a change to use the full CPU power.

When dealing with function calls, having large string parameters (for example file contents, returned by ReadFile), is better to send the parameter by reference, to avoid copying the string. This method can be used only if the called function doesn't alter the string contents.

For example:

```
[..]
    foo(string filecontents) {
        // do something with filecontents
    }

    testFoo() {
        var data = ReadFile("somebigfile.bin");
        foo(data);
    }
[..]
```

The previous example will cause the *filecontents* parameter to contain a copy of the file content. For faster execution, you can send the file contents by reference:

```
foo(var string filecontent) {
    // do something with filecontents
}
```

Note the **var** added to the parameter. This method is convenient only for large string parameters, because arrays, delegates and objects are sent by reference by default (object reference, not variable reference).

14.2 Optimize memory usage

When creating CAS applications, the memory usage may be a concern, when dealing with a great amount of concurrent users. For this, Concept Core has a few optimizing mechanisms, like shared memory for the byte code and variable memory allocation at first usage. In other words, a variable will be allocated on first use only. This reduces the amount of memory used by Concept Core. Also, by default, Concept Core doesn't impose limits on the used memory, in theory the only limit is given by the hardware.

Each concept variable uses additional overhead information. Each variable uses at least 15 bytes on 32 bit architectures and 19 bytes on 64-bit architectures. Arrays, objects and strings may use additional meta data. The variable meta data keeps informations like variable type, link count and property data. When using arrays, each array element is a variable. An array of 1024 elements will use at least 15kB on 32 bit and at least 19kB on 64 bit architectures. A as further optimization, an array element is created when first used. For example if *arr* is an empty array, and *arr[1000] = 1*, then the array will use just the memory needed for a single variable. The length of the array will be 1001. The other elements will be allocated when first used. Arrays use some additional space for meta-data keys and elements.

The same happens with class objects. Each member variable will be allocated on first use. If a member is never used, it will never be allocated.

Variable meta data is useful to the Concept Core to manage the memory usage and variable types. For example, when a variable's link count reaches 0, it will automatically be freed.

The only thing to avoid are circular references, however, the Concept

Garbage Collector is able to detect them. For example, is better to avoid situations like:

```
this.SomeMember.SomeOtherMember = this;
```

This will generate a cyclic reference and will require some additional attention from the Concept Core. It is recommended to avoid these situations.

Concept Core manage its own memory. However, when using low level APIs (static C functions), require explicit memory management. For example, using the *TCPSocket* class doesn't require attention to memory allocation or closing the socket before TCPSocket is freed. If low-level *standard.net.socket* APIs are used, explicit call to *SocketClose* is mandatory for avoiding memory leaks.

Starting Concept 4.0, for UI application only, a *Workers* parameter may be set in the application manifest. When set to 1, all clients will run in the same process, minimizing the memory usage (code and static libraries will be loaded once for all the user).

test.con.manifest

```
[Application]
Workers =  1
```

If Workers is set to a higher value than one, it will represent the number of users per process. For example, if set to 100, each process will be responsible for 100 concurrent users.

Setting this parameter will make the server to scale almost linearly (see figures 14.1 and 14.2).

The only disadvantages of using shared processes is if a instances crashes, will disconnect all the users connected to the process.

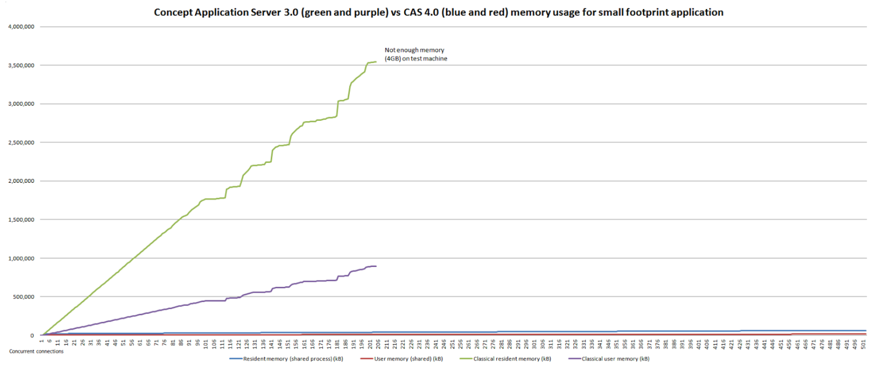


Figure 14.1:
Scalability for small applications (Concept 3 vs 4 with Workers = 1)

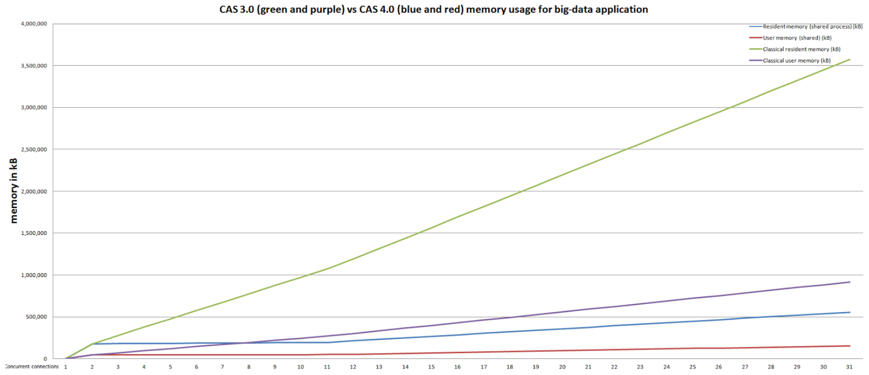


Figure 14.2:
Scalability for big applications (Concept 3 vs 4 with Workers = 1)

14.3 JIT-friendly code

Concept Core uses a JIT(Just-In-Time) compiler for running native code on the CAS platform. It currently generate code for Intel x86-32, AMD x86-64, ARM (including ARM-v5, ARM-v7 and Thumb2 instruction sets), IBM PowerPC-32, IBM PowerPC-64, MIPS-32 and SPARC-32. The JIT is based on the *sljit* project (stack-less platform independent JIT compiler) with various code optimizations attached, like array initialization loop detector, arithmetic simplification, and pipeline stall optimization.

Every Concept function is compiled on the second call, because compilation, run-time optimization and memory overhead are considered more expensive that a single function call. By using this method, only the frequent called functions are compiled, resulting in an optimal balance between speed and memory overhead.

The JIT compiler analyzes the CAL byte code, and creates multiple entry points on the same function. When an operation that cannot be handled by the native code is requested, the JIT code simply exits to the Concept Core Interpreter, that will continue the execution from that point to the next JIT block.

For now, object creation, delegate calls, static function calls, object member access and Concept function calls are not handled by the JIT code, some string and array operators and the **echo** and **throw** keywords, falling back to the Concept Interpreter. After these unsupported operations are executed, the Concept Core Interpreter will switch back to JIT-generate native code.

All numerical operations are executed by the JIT. Note that a division by 0 run-time error, reported by the Concept Core Interpreter, will not be reported by the JIT native code, resulting in evaluating the response to $+\infty$ or $-\infty$ (or the maximum/minimum double precision floating point value).

With ideal code, the Core uses almost exclusively the JIT. In practice, reading members, calling functions or creating objects, will break the JIT code.

For example:

```
for (var i = 0; i < 100000; i++)
   this.SomeMember += i;
```

Will use Concept Interpreter for every operation on *this.SomeMember*. The JIT-friendly way is:

```
var a = this.SomeMember;
for (var i = 0; i < 100000; i++)
   a += i;
this.SomeMember = a;
```

The previous example is faster on the Concept Core Interpreter, and the loop will also use JIT-generated uninterrupted binary code.

Loops calling the same function on the same parameters should be avoided, for example:

```
import standard.lib.math
[..]
var a = 0;
for (var i = 0; i < 100000; i++)
   a += i * sin(M_PI/2);
```

This will break the JIT loop on every iteration. The correct way of doing this is:

```
var sin_pi_2 = sin(M_PI/2);
for (var i = 0; i < 100000; i++)
   a = i * sin_pi_2;
```

On this implementation, the loop will run interrupted.

The JIT compiler detects some special situations. For example, if an array needs to be initialized, is better to do that in a separate loop.

The unfriendly example.

```
var[] arr;
var a = 0;
for (var i = 0; i < 100; i++) {
   arr[i] = 1;
   a += i;
}
```

The previous example will be executed relatively fast, but it can go even faster, if the loop is broken into two loops:

```
var[] arr;
var a = 0;
// JIT will detect this array
// initialization
for (var i = 0; i < 100; i++)
   arr[i] = 1;
```

```
for (var i = 0; i < 100; i++)
    a+= i;
```

When using this method, the JIT optimizer will detect the array initialization, and execute the loop in a single step.

Loops increments are very important to JIT. If it detects that a loop uses a locally variable, that is not used as parameter, result or divide/modulus operations, will assume that it has to deal with an integer value, and instead of using double floating points as iterator value, it will use integer values, which are significantly faster. Note that the increment must be done via the ++ operator for safe detection.

For now, the only bottleneck in the JIT code is the use of the floating point registers, executing slower when compared to integer values. In the next CAS versions, more integer detection optimizations will be added, reducing the gap.

When dealing with function consuming lots of CPU cycles, you may want to force the JIT compilation from the first run. This may be done by simply adding a dummy call to the function. For example:

```
    do_sum(number start, number end) {
        var result;
        for (var i = start; i <= end; i++)
            result += i;
        return result;
    }

    Main() {
        // a dummy call
        do_sum(0,0);
        // the second call will be run natively
        echo do_sum(0, 1000000);
    }
```

The first call to *do_sum* does nothing. It is just a dummy call, to force the function call count to increment. The second call will force the compilation of the function, resulting in full native machine code.

Automating type conversion should be avoided. An automatic conversion from string to number will result in JIT break and fall back to Concept Core Interpreter. As a “bad” example:

```
1 // evaluate 1 + value "2"
2 var a = 1 + "2";
3 a *= 2;
4 a += "3";
```

Line 2 and 4 will always cause a JIT break, because the core will need to evaluate “2” to number 2, and “3” to number 3.

14.4 Profiler

A software profiler is a tool that allows the developer to analyze the program in run-time. It will collect data regarding functions or exceptions (or other more specific instructions). It will help in program optimization, by identifying bottlenecks.

Concept core uses a simple profiler that will analyze the function execution time and exceptions. For profiling an application, the developer must simply instantiate the *Profiler* class, defined in *Profiler.con*. Note that Concept profiler cannot analyze code executed on GPU via *__gpu* or *__parallel* macros.

The profiling data can be read by using one or more of the following properties:

Call

Returns an unsorted array having class and function (in the form *class.functionname*) name as key and call count as value.

Duration

Returns an unsorted array having class and function (in the form *class.functionname*) name as key and total call duration as value. If a function is called multiple times, it will hold the summed execution time.

Throw

Returns an unsorted array having class and function (in the form *class.functionname*) name as key and the throw statement reached count as value.

CallSorted
Sorted version of *Call*, returning instead an array composed of arrays with two elements. The first element in the child array is the function name and the second one is the call count.

DurationSorted
Sorted version of *Duration*, returning instead an array composed of arrays with two elements. The first element in the child array is the function name and the second one is the duration.

ThrowSorted
Sorted version of *Throw*, returning instead an array composed of arrays with two elements. The first element in the child array is the function name and the second one is the reached *throw* count.

No external tool is needed for analyzing the content. A simple console example, will use the following code:

```
include Profiler.con
import standard.lib.thread

class Main {
    foo() {
        return 1;
    }

    foo2() {
        Sleep(100);
        return 0;
    }

    Main() {
        // create the profiler
        var p=new Profiler();

        foo2();
        foo2();
        foo();

        // show collected data
        echo p.DurationSorted;
    }
}
```

The output should look like:

```
Array {
    [0] =>
        Array {
            [0] => Main.foo2
            [1] => 218.4
        }
    [1] =>
        Array {
            [0] => Profiler.GetDurationSorted
            [1] => 0
        }
    [2] =>
        Array {
            [0] => Main.foo
            [1] => 0
        }
}
```

For *concept://* applications, the *ProfilerForm* helper class, defined in *ProfilerForm.con* may offer a convenient method of getting the profiling data. The constructors takes two parameters:

```
ProfilerForm(Owner, Profiler prof);
```

Where *prof* is a *Profiler* object and *Owner* is the application main form (or any other form). It doesn't matter who owns the *ProfilerForm*, the entire program being analyzed.

A minimal GUI example would be:

```
include Application.con
include ProfilerForm.con

class MyForm extends RForm {
    MyForm(Owner) {
        super(Owner);
        var profiler = new ProfilerForm(this, new Profiler());
        profiler.Show();
    }
}

class Main {
    Main() {
            try {
                var Application = new CApplication(new MyForm(null));
                Application.Init();
                Application.Run();
                Application.Done();
            } catch (var Exception) {
                echo Exception;
            }
    }
}
```

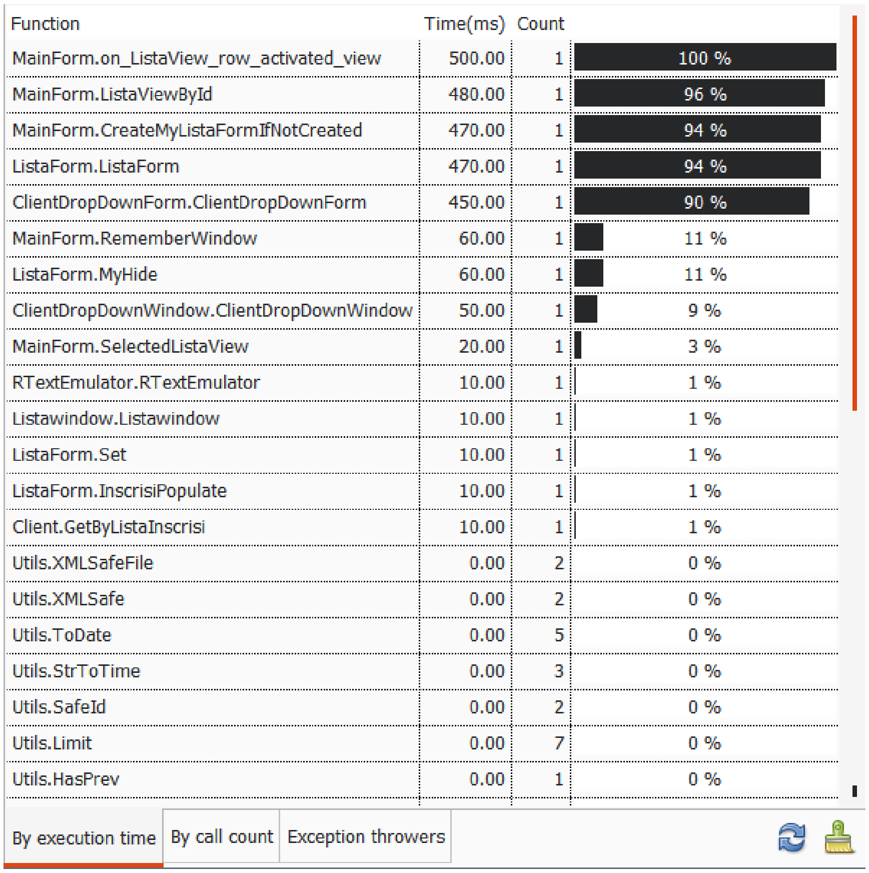
This will actually produce no profiling data (no code is executed after the creation of the *ProfilerForm*).

However, for a complex application, the output may look like in the figure 14.3.

The *ProfilerForm* will refresh its data every 2 seconds, unless the *RefreshTimeout*(milliseconds) property is set to another value (default is 2000).

14.5 Comparison with other platforms

The Concept Programming Language was designed to be easy to use, with an rapid learning curve, with syntax familiar for most programmers (similar to C++, C#, Java, JavaScript). The Core is designed to be both fast and memory efficient, for running on a server environment with minimal overhead. Being a hybrid core (using both an interpreter and a JIT compiler), makes it hard to compare with both interpreters and compilers. As an interpreter, Concept is reasonable fast (with speeds slightly higher than PHP or Python). The JIT core is about 10 times faster overall than the interpreter core. On arithmetic operations the JIT



Function	Time(ms)	Count	
MainForm.on_ListaView_row_activated_view	500.00	1	100 %
MainForm.ListaViewById	480.00	1	96 %
MainForm.CreateMyListaFormIfNotCreated	470.00	1	94 %
ListaForm.ListaForm	470.00	1	94 %
ClientDropDownForm.ClientDropDownForm	450.00	1	90 %
MainForm.RememberWindow	60.00	1	11 %
ListaForm.MyHide	60.00	1	11 %
ClientDropDownWindow.ClientDropDownWindow	50.00	1	9 %
MainForm.SelectedListaView	20.00	1	3 %
RTextEmulator.RTextEmulator	10.00	1	1 %
Listawindow.Listawindow	10.00	1	1 %
ListaForm.Set	10.00	1	1 %
ListaForm.InscrisiPopulate	10.00	1	1 %
Client.GetByListaInscrisi	10.00	1	1 %
Utils.XMLSafeFile	0.00	2	0 %
Utils.XMLSafe	0.00	2	0 %
Utils.ToDate	0.00	5	0 %
Utils.StrToTime	0.00	3	0 %
Utils.SafeId	0.00	2	0 %
Utils.Limit	0.00	7	0 %
Utils.HasPrev	0.00	1	0 %

Figure 14.3: ProfilerForm

is about 20 times faster. In this case, is comparable in speed with binary code generated by the C/C++ compilers.

A popular choice is the sieve of Eratosthenes benchmark. In mathematics, the sieve of Eratosthenes, one of a number of prime number sieves, is a simple, ancient algorithm for finding all prime numbers up to any given limit. It does so by iteratively marking as composite (i.e. not prime) the multiples of each prime, starting with the multiples of 2[1].

The used program was:

```
import standard.C.time
```

[1] Source: http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes as of January 24, 2014

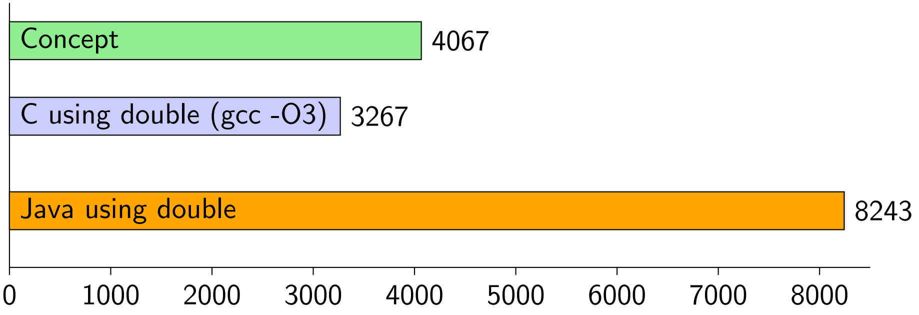
```
import standard.C.math

class Main {
    Sieve(SIZE, timeout) {
        if (!SIZE)
            return;
        var[] flags;
        var i, prime, k, iter, count;
        var iterations, seconds, score, startTime, elapsedTime;

        startTime = clock();
        do {
            count = 0;
            for (i=0; i<SIZE; i++)
                flags[i] = true;

            for (i=0; i<SIZE; i++) {
                if (flags[i]) {
                    prime = i + i + 3;
                    for (k=i+prime; k < SIZE; k+=prime)
                        flags[k]=false;
                    count++;
                }
            }
            iterations++;
            elapsedTime = clock() - startTime;
        } while (elapsedTime < timeout);

        seconds = elapsedTime/1000;
        score = round(iterations/seconds);
        echo "$iterations iterations in $seconds seconds (score: $score, count: $count)\n";
    }

    Main() {
        // force the JIT
        Sieve(0, 0);
        Sieve(8192, 10000);
    }
}
```

The same program was translated in C, Java and PHP 5.4.

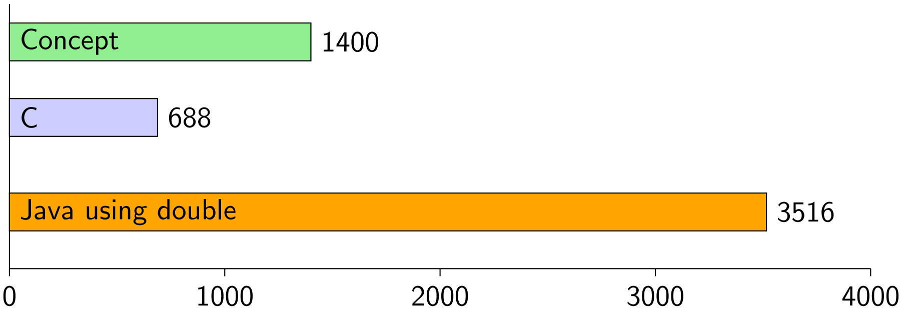
The results were (higher is better):

Language	**Score**	**Used memory**	**Notes**
Java, using double	8243	3,516 kB	
C, using int	7435	688 kB	gcc -O0 (no optimization)
Concept	4064	1,240 kB	
C, using double	3267	688 kB	gcc -O3 (max optimization)
PHP	393	3,180 kB	

The results for the sieve benchmark implementation(using doubles).



Sieve score chart (higher, the better)



Memory usage chart (lower, the better)

When using double floating point numbers, the C compiler(gcc) with maximum level of optimizations produces code slower than Concept (see first chart). Java however, on the same test is twice as fast. When it comes to memory, for the same test, C is the lightest, using about half the memory used by Concept. This is, because Concept stores meta-data with every variable. Java however, uses twice the memory used by Concept. This could be regarded as a good speed - memory usage ratio. Note that hundreds of benchmark, on different operations can be made. The Sieve benchmark gives no absolute comparison between execution time, just some hints.

A second test, was to find all the prime numbers smaller than 10,000,000. This test is intended to use big arrays. The test function was:

```
get_primes7(n) {
    var s = new [];
    for (var i = 3; i <= n; i += 2)
        s[length s] = i;

    var mroot = floor(sqrt(n));
    var half = length s;
    i = 0;
    var m = 3;

    while (m <= mroot) {
        if (s[i]) {
            var j = floor((m*m-3)/2);
            s[j] = 0;
            while (j < half) {
                s[j] = 0;
                j += m;
            }
        }
        i++;
        m = 2*i + 3;
    }

    var res = [ 2 ];
    var len = length s;
    for (var x = 0; x < len; x++) {
        if (s[x])
            res[length res] = s[x];
    }
    return res;
}
```

This function is difficult for Concept Core, because it has multiple JIT breaks. It is executed partially by JIT, and partially by the interpreter core.

The results were (lower time, the better):

Language	**Time(ms)**	**Used memory**
C++ (with std::vector)	546	25 MB
Concept	2577	127 MB
PHP	6421	488 MB

This test clearly shows the difference in memory usage between C++, with homogeneous arrays (std::vector<int>) and Concept heterogeneous arrays. PHP also uses heterogeneous arrays, being included in the benchmark. For this test, PHP 5.4 command-line interpreter uses over almost 4 times the memory used by Concept, and executes in over twice the time.

As expected, C++ homogeneous arrays are faster and uses far less memory (with no meta-data needed).

Concluding, Concept is slower than Java, slightly faster than C/C++ when dealing with floating-point numbers and faster than most interpreters.

The language itself, is weak typed, and the core automatically manages the memory, making Concept easier to use than C/C++ or Java. CAS, comes by default with everything needed for development and deployment and a rich framework with about 200 high level classes, with easy to use interfaces, and over 7,000 low-level APIs. The entire framework uses straight forward and intuitive interfaces. No pointers or memory addresses are managed by the programmer.

As a disadvantage when comparing with strong typed languages, is that not all the errors can be identified at compile type. For example, if a function taking an untyped parameter, assumed to be an array, receives a number, a run-time error will be generated when the number will use an array-specific operator, like [ ]. For strong-typed languages, this situation can be identified at compile time.

The code written in Concept and compiled, will run on any CAS, regardless the operating system or architecture. The Concept Client allows you to run instances of Concept applications on most devices, starting from desktop computers to tablets, smart phones, smart TVs and embedded systems.

Part IV

Real applications

Chapter 15

Problems and solutions

A technology, platform, programming language or database server must be chosen for the right reasons. Concept is a great platform for cloud-based, model-driven applications. It offers you about 80 to 90% of any application, by providing both the Concept Framework and development tools like CIDE and GyroGears.

Let's define 'the customer" as the one having a problem, the "solver" is you or the analyst, the "end-user" is the user of the system (part of "the customer") and the "client" as the customer's client. Note that in a few cases, the customer may be the end-user.

Usually "the customer" identifies or anticipates a problem within his organization and contacts the "solver". Before describing the problem, you must never let the customer describe the solution. The main task for the customer is the problem description and nothing more. When the customer specifies a solution, you're on a direct road to a failed solution. The customer usually lacks the knowledge, the vision, or simply misses the outside perspective. It will tend to focus on it's own perspective, but usually, the end-users are the one inputing, working, processing, extracting and reporting. Just think at this: when you go to the dentist, you tell the dentist to pull your tooth out, or what it tooth hurts you? (assuming that you don't do regular check-ups). This is the case here. The customer never knows what is best for his organization and is best to focus on the problem only.

Note, that the customer may not see or correctly identify a problem. The solver must help the customer with problem identification, by analyzing his organization processes. This usually comes in time, with experience. There will always be unforeseen problems and the solver should not be afraid of that. The solver should be working in a few steps, not necessary in the given order:

Step 1
Isolate the output data, documents, forms, reports or interfaces (for example when interfacing an physical equipment)

Step 2
Collect all the documents, forms or data models. Get real samples, ideally following the data from start to end.

Step 3
Analyze the work flows. Identify each end-user's role.

Step 4
Identify the contact persons. Usually the number of contacts must be to the minimum, with an ideal of 1 responsible. This is useful for forcing the customer to centralize communication and discuss internally the problem in internal meetings, and provide only the conclusions.

Step 5
Define the problem. Then split the problem in the maximum number of sub-problems. Let's call the sub-problem an "atomic problem", meaning a problem that cannot be split anymore.

Pay attention to the "end-user", as it may often be a weak point in the solution. It's better to minimize the solution's impact on the end-user, by including a high degree of automation. Even the end-user education or personality can have an impact on the solution.

The word "software" cannot be used yet. The solver must now define the problem. This is the most important step, because an well-defined problem is as solved. For example, if the customer has difficulties storing, accessing and sharing documents in his organization, it may ask for a document management solution. A software seller will guide the customer to a generic document management system (DMS), but keep in mind that it won't be a perfect fit. However, if "the solver" helps the customer to better describe the problem, and then tries to find a "best fit" solution, supporting the actual work flows, rather than forcing the organization to adapt to a generic solution, then it will actually solve the problem, rather than minimize it. For example, the customer may have generic forms, documents that will need some data collecting and reporting.

Assuming that the customer and the solver defined the problem, and then, the solved split the problem to atomic problems, we can now talk about "the software" and generate a list of requirements, for the best decision when it comes of technologies or database servers.

Behind the "solver" may be a big team, or it may just be a freelancer, doing both the analysis and the implementation. We will only focus on the "customer" and the "solver" tasks. After the atomic problem list is created, the solver must create a list of priorities and condition. For example, a problem may not be solved before another one is solved. In some situation, the solver may want to identify the minimum subset of atomic problems that will provide a partial working solution. At this point, the solution starts becoming a software system. In this case, the customer may benefit a partial solution before is fully implemented, reducing the implementation time.

Partial solutions are dangerous for the customer and the end-user. The customer may have tendencies to intervene in the development process, leading to a failure. Also, the end-user may be reluctant to cope with frequent changes or updates, especially when it comes to procedure modifications between updates. Partial updates are only good when dealing with mature customers and/or when it may increase productivity.

The solver is responsible for the customers success or bankruptcy as it may identify problems in the customer's procedures. This being the case, it must discuss and clarify before the implementation. It must never hide a procedural problem to the customer. It may never ask the customer to change a working procedure, because it would be easier for the solution. The solution should always fit the client, not the other way.

It is important for the solver to avoid showing anger, dislike, disapproval, even if the situation is tense. It must be very communicative, always sustain any decision, keep the customer informed, be sincere, and resist to any kind of pressure. His or hers job is to solve a problem, and a professional solver has a sole objective: a solution. The solver must solve problems, not work for being liked by the customer or the end-user, so keep a cool head. A solver needs tons of patience and optimism, being a pessimist will make your job twice as hard.

A successful solver will not need nor accept dead-lines. Dead-lines are toxic to complex projects, that usually will be in a state of permanent evolution. A software system will only be over when it will have no more users. As a solver, I prefer to tell the customer that it will be ready when the problem will be solved. If you have enough experience with similar projects, you may give a general time frame. However, even with clients in the same vertical market, the solver will encounter different problems.

The solver must never be a seller. It must not negotiate with the client, promote or impose actions strictly for its own profit.

15.1 HR application basics

Human resource management (HRM, or simply HR) is the management process of an organization's workforce, or human resources. It is responsible for the attraction, selection, training, assessment, and rewarding of employees, while also overseeing organizational leadership and culture and ensuring compliance with employment and labor laws[1].

We will use the basic recruiting process as an example. First of all, the HR staff needs some CV sources, like web sites, e-mails or event social media, for example LinkedIn. It also makes headhunting operations. A number of companies, especially the one in production take CVs from skilled workers at the factory's gate. Those CVs are usually pre-printed forms, on which the candidate just checks some boxes.

A typical CV comes usually in a standard form. For example, in Europe there is a popular "European" CV format. It includes even the candidate's picture, especially useful when hiring a secretary for the top management. The problem arrives from different CV/Resume formats, all of the containing the same information, just organized in different forms.

A solver must identify at least some basic fields in the CV, like name, date

[1] http://en.wikipedia.org/wiki/Human_resource_management on January 27, 2014

of birth/age, sex, contact data, education, skills and experience. This is the basic data that needs to be extracted for free-form CVs.

The CV may be received in various formats. For example, PDF, HTML (when downloaded from a website), a scanned document or even as TXT. The solver must create an independent procedure for extracting data from a CV, regardless its format. It must create a function similar to this:

```
import standard.lib.poppler
import standard.lib.ocr
import standard.C.io
include HTMLTrimmer.con
include DocX.con

class Main {
    ExtractPDFText(string filename}) {
        var pdf = PDFLoad(ReadFile(filename), "", var err);
        var res = "";
        if (pdf) {
            var pages = PDFPageCount(pdf);
            for (var i = 0; i < pages; i++) {
                res += PDFPageText(pdf, i);
                res += "\n";
            }
            PDFClose(pdf);
        } else
            throw "Error opening pdf file ($err)";
        return res;
    }

    GetCVAsText(string filename) {
        var ext = ToLower(Ext(filename));
        switch (ext) {
            case "pdf":
                // no break needed here, because
                // the function will return
                return ExtractPDFText(contents);
            case "png":
            case "bmp":
            case "jpeg":
            case "jpg":
            case "tiff":
                if (!OCR(filename, var data))
                    return data;
                throw "Error performing OCR";
            case "htm":
            case "html":
                return HTMLTrimmer::(ReadFile(filename), 0xFFFFFF,
                    0xFFFF, true);
            case "docx":
                return DocX::GetText(filename);
            case "txt";
                return ReadFile(filename);
        }
        throw "Unsupported format";
    }

    Main() {
        var text = GetCVAsText("CVExample.pdf");
        // now in text we have the actual text in the cv,
        // independent of the given format (it may be pdf, scanned
        // document or plain text)
    }
}
```

Using the *HTMLTrimmer* the solver could load CV's in HTML format and extract the text from there. The HTML trimmer class enables the programmer to cut HTML text to a maximum of length. It also enables the programmer to extract plain text from HTML documents by using the *HTMLTrimmer::Go* function:

```
static Go(string html, number max_len, objects=-2,
    return_plain_text=false);
```

The *max_len* parameter, specifies the maximul length of the text contained by the returned result. If result is not plain text, the tags will automatically be closed, ensuring the HTML content remains valid. *objects* is the maximum number of images before the HTML content gets cut. In our case, the function is only used for converting HTML to text, by setting the *return_plain_text* parameter to true.

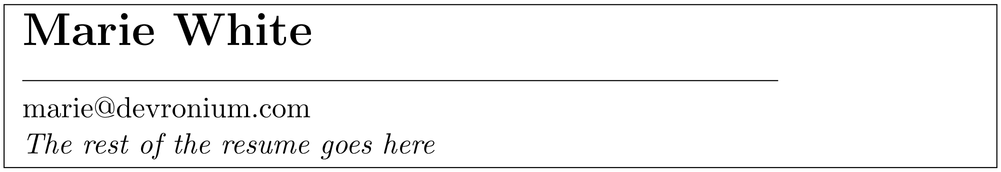
The *DocX* class, allows reading contents from a *.docx* file as plain text. Docx files are simply zip archives containing some XML files. The *DocX* class opens the .docx files as archives, identifies the content XMLs and performs a transformation, using an XSL template, to plain text.

This step is called normalization, where documents if various formats are converted to a format that the system will be able to analyze.

After the normalization, the system must correctly identify phone numbers and e-mail addresses. For contact information, the candidate will not necessary have a trigger word, for example, “e-mail: maria@devronium.com”. It may just add an e-mail address under his or hers name. The same goes for phone numbers. For this, the regular expressions are great.

```
import standard.C.io
import standard.lib.preg
[..]

class Main {
   [..]

   Main() {
       var text = GetCVAsText("CVExample.pdf");
       var email_arr=preg(text,
           "[A-Za-z0-9_\\.\\-]+\\@[A-Za-z0-9_\\.\\-]+\\.[A-Za-z0-9_\\.\\-]+");
       echo email_arr;
   }
}
```

Assuming that the *CVExample.pdf* has a structure similar to:



Marie White

marie@devronium.com
The rest of the resume goes here

The program will return [“marie@devronium.com”]. When a CV will contain multiple e-mail addresses, each of the them will be contained in the returned array. The use of regular expressions for identifying data is great for free-form documents.

For identifying the name, the task may be a little difficult. Usually is the first line in a CV, but the solver should exclude noise words like *Curriculum Vitae* or *Resume*.

After correctly identify the basic data in a CV, the system will load standard fields in a database, for easy searching. However, an end-user will want to perform natural language searches. For example, “engineers willing to relocate with good spoken English”. For this, Xapian will be just perfect. Every text blob returned by *GetCVAsText* may be indexed by Xapian, and the end-user may perform the searches in natural language.

Note that the difference between a great application and a mediocre one is the search. In most data-driven applications, the search is essential. SQL queries are NOT searches. For example, *“select * from cvs where keywords like ’%engineers willing to relocate with good spoken English%’”* it will not return any data. Also *select..where* statements will never perform any raking of the results. Probabilistic search engines, like Xapian, rank items based on measures of similarity (between each item and the query, typically on a scale of 1 to 0, 1 being most similar) and sometimes popularity or authority or use relevance feedback[2]. As a secondary advantage of using true search in an application, is the speed. Probabilistic search engines are incredibly fast, being able to look in millions of documents in under half a second.

The solver has created now a method of importing a CV into a database, a procedure for searching the data. Similar to the CRM application, the HR staff needs to be able to add comments, statuses and schedule interviews. An interview may be by phone or face to face.

The recruiters must be able to create shortlist, containing selected candidates. Each candidate may be tested. It may have to pass psychological or aptitudes tests. Different candidate markers may be used.

The solver will use the following entities:

Candidates
  the actual candidate list (the CV database)

Positions
  the open positions

Employees
  the employees

[2] Source: http://en.wikipedia.org/wiki/Search_engine_(computing) on January 27, 2014

Candidates will be

Interviewed
the schedule and actual interview feedback

Selected or declined
if selected, the candidate will be added to a shortlist

Tested
the candidate will be tested, if necessary

For each position, a shortlist will be created, containing a few recommended candidates for the given position. Then the recruiters will do their job, one or more of the candidates, with the specialized department from the company. For example, the HR gals will create the shortlist for an open position in engineering. With the engineering department, will hire a candidate or more. If a candidate is not hired, it will have all the history in the system, for later openings. The HR department may also need black list, for candidates that will not be contacted for any other opening.

The solver may add some performance or statistical reports. For example, from the total of seen candidates, how many were hired.

Some communications methods may be added, for example, an automatic mailing keeping the candidates informed with openings. Alternatively, SMS gateways or automated calls may be used.

A solver may over-engineer a HR application by adding automatic recommendations, for a given position, based on the data collected from the HR end-users. For example, if an user performs some searches for the opening in “Software development”, and checks out some CVs, the system will record its behavior, and then use the slope one algorithm to generate recommendations for others, or for the user itself.

Slope One is a family of algorithms used for collaborative filtering, introduced in a 2005 paper by Daniel Lemire and Anna Maclachlan. Arguably, it is the simplest form of non-trivial item-based collaborative filtering based on ratings. Their simplicity makes it especially easy to implement them efficiently while their accuracy is often on par with more complicated and computationally expensive algorithms[3].

[3] http://en.wikipedia.org/wiki/Slope_One on January 27, 2014

For using slope one recommender, the solver may encode the end-user's action to different values. For example, a black-listed candidate will be rated with 0, an opened CV will be 1, a contacted candidate will be rated with 2, an interviewed candidate with 3, a tested candidate with 4, a candidate that maded to the short list is 5 and a hired candidate is 6 (though it will not be used in our example). For better understanding an algorithm, will use a minimalistic set of open positions and candidates.

Position	Marie White	Jack Black	Eddie Brown
Software developer	5	3	1
Analyst	2	1	4
Software tester	3	1	**?**

Let's assume that an end-user scheduled an interview with Marie's, after seeing her resume, and then looked at Jack's resume. The system now may calculate what the user will do with Eddie.

We see that for the software developer position, Eddie was rated lower than Marie by 4 and lower than Jack by 2. For the analyst position, Eddie was rated higher than Marie by 2 and higher than Jack by 3. This means that on the average, Eddie is rated $\frac{-4+2}{2} = -1$, in relation with Marie, and $\frac{-2+3}{2} = 0.5$ in relation with Jack.

For the software tester, Eddie, should score 2 (3 - 1) in relation with Marie, and 1.5 (1 + 0.5) in relation with Jack. In the end, the Eddie will most likely score 1.75, from computing:

$$\frac{2 \times Marie_{rc} + 1.5 \times Jack_{rc}}{Marie_{rc} + Jack_{rc}}$$

Where $Marie_{rc}$ is the ratings count for Marie (2, in our case, one for Software devloper, and one for Analyst), and $Jack_{rc}$ is the ratings count for Jack.

This means that Eddie is most likely to be contacted by the user or at least seen (1.75 may be rounded to 2). It is a very simple algorithm with almost magical impact to the end-user. It may be used on any other application, needing a form of automated recommendation to the end-user.

I'm a big fan on these kind of optimization, and I recommend it to you too. For up to a few hours of extra work, the application may be a pleasant surprise to the end-user, and a happy end-user, means greater chances of success in the implementation.

15.2 Telco billing basics

A few years ago, I've implemented a CRM for a telecommunication company. It was a fun project, because it dealt with a small but growing local company. It provided both telephony and Internet access services. Their problem arrived from the fact that the billing was done almost by hand, by using an in-house billing software, in which some text exports from the softswitch were loaded and then parsed. A softswitch, short for software switch, is a central device in a telecommunications network which connects telephone calls from one phone line to another, across a telecommunication network or the public Internet, entirely by means of software running on a general-purpose computer system. Most landline calls are routed by purpose-built electronic hardware; however, soft switches using general purpose servers and VoIP technology are becoming more popular[4]. The software work great, the only problem it was that it needed monthly inputs, while synchronizing commercial data with the billing system. They wanted the unification of the commercial data, with billing, technical support, equipment management, and accounting software integration.

At that time, the ported numbers were a bit of a problem, because of different cost per minute for different operators. Also, their clients had a few subscriptions and extra options, with different quantum pricing. For example, if a client used the phone for 40 seconds, a minute was charged. For other subscription, only the calls lasting for less than a minute were rounded to a minute. Another service is the toll-free telephone number, which is free for the calling party, and instead the telephone carrier charges the called party for the cost of the call.

The Internet service billing was straight forward, since it were only fixed subscriptions. The system automatically managed the bandwidth, according to the client's subscription.

Another job for the system was to suspend the services for the debtors or

[4] http://en.wikipedia.org/wiki/Softswitch on January 27, 2014

for security reasons. For example, a subsystem analyzed all the calls for a specific client, and if it detected a high volume of international calls, in a short period, originating from the same number, it suspected an attack on the client and generated some alarms.

The billing sub-system uses only the following entities:

The client
identifying the client with billing and service addresses

The contract
identifying the client's contract(s)

The subscription(s)
the client subscriptions for a given contract

The allocated phone number(s)/IPs/Username
the service specific data

A loop was created for real-time pre-processing of the telephony data. It identified the caller phone number, the callee phone number, the route used, duration and checked the network of origin and destination, checking also with the ported number database to see if a number was ported at the date of the call. The loop loaded all the ported number into the RAM, in an array (key-value). I decided to load everything in memory, for speed, because additional queries may take some time and generate some overhead. At the time, there were under 1,000,000 ported numbers, so keeping them in RAM used about 100MB of RAM. This is reasonable for the task, because the loop can check very fast if a number is ported or not.

The resulting structure and data:

Timestamp	**Caller**	**Calee**	**Caller network**	**Calee network**	**Route**	**Duration**
2014-01-21 10:23:21	0370481231	0722541001	BeeVoice	Vodafone	Vodafone	350
2014-01-21 10:23:40	0370480131	0745582759	BeeVoice	Orange	Orange	132
2014-01-21 10:23:43	0758387592	0370489980	Orange	BeeVoice	Orange	967

For the toll-free numbers, the Caller and Callee were inverted. This was done this way for simplifying the analysis when the bill was generated.

The loop pseudo-code:

```
[..]
class CallData {
    var Timestamp;
    var Caller;
    var Callee;
    var Caller_network;
    var Callee_network;
    var Route;
    var Duration;

    Write() {
        // write to a database
        [..];
    }
}

class Main {
    [..]

    Main() {
        var ported = LoadPortedNumbers();
        while (true) {
            var call_data = WaitForDataFromTheSoftwSwitch();
            if (call_data) {
                var ported_network = ported[call_data.Callee];
                if (ported_network)
                    call_data.Callee_network = ported_network;

                ported_network = ported[call_data.Caller];
                if (ported_network)
                    call_data.Caller_network = ported_network;

                call_data.Write();
            }
        }
    }
}
```

When the invoices were generated, the system just analyzed the meta data, already recorded by the loop, and applied the client's subscription. Some clients had options with minutes included in the subscription cost. On that case, the system charged only the minutes overcoming the included minutes. The generated invoices are automatically sent by e-mail to the client.

The system keeps track of the invoice collection, and after the given payment term expired, it automatically blocked the client's access to Internet and telephone services.

The billing loop pseudo-code:

```
[..]
class Main {
    [..]

    ProcessPhoneNumber(phonenumber, invoice, service) {
        while (var call_data = FetchPhoneEvent(phonenumber)) {
            var duration = call_data.Duration;
            // if call is less than 60 seconds, get it rounded to
            // 60 seconds
            if (duration < service.MinQuantum)
                duration = service.MinQuantum;

            // assuming that for service.Plan is the subscription
            // plan, for each outgoing network
            var call_price = service.Plan[call_data.Calee_network] *
                duration/60;

            invoice.Phone_calls_cost += call_price;
        }
    }

    ProcessContract(contract) {
        var client = contract.Client;
        var services = contract.Services;
        var len = length services;

        var invoice = new Invoice();
        invoice.Services = services;

        for (var i = 0; i < len; i++) {
            var service = services[i];
            invoice.Total += service.Value;
            invoice.Tax += service.Tax;

            var phonenumbers = service.Phonenumbers;
            if (phonenumbers) {
                var len2 = length phonenumbers;
                for (var j = 0; j < len2; j++) {
                    var phonenumber = phonenumbers[]
                    ProcessPhoneNumber(phonenumber, invoice, service);
                }
            }
        }

        invoice.GenerateDocument();
        invoice.Write();
    }

    Main() {
        var contracts = GetContracts();
        var len = length contracts;
        for (var i = 0; i < len; i++) {
            var contract = contracts[i];
            ProcessContract(contract);
        }
    }
}
```

Note that in every complex system, simplicity is what makes it work. Just remember to split the problems again and again until left only with atomic problems. Also, try to limit the amount of processing per function, not for technical but rather for aesthetically reasons. It will be easier to read.

For the Internet connection access, a RADIUS server was used. Remote Authentication Dial In User Service (RADIUS) is a networking protocol that provides centralized Authentication, Authorization, and Accounting (AAA) management for users that connect and use a network service[5].

The used RADIUS server is FreeRADIUS, an open source project. Their setup used a MySQL database for storing the users, and making the activation/deactivation of an user trivial (plain SQL insert/update and deletes).

For other equipments, the SNMP protocols and RouterOS APIs were used to control the network access. Some clients don't use PPPoE accounts and are to be suspended/reactivated from specific equipments using proprietary protocols.

[5] http://en.wikipedia.org/wiki/RADIUS on January 27, 2014

15.3 User tracking application

Some time it will be needed to track an user. For this, two methods are available. The first is to track the user after the IP. This provides localization at the city level. The second one, enables you to read the location of the user using the GPS or cell towers. This method is only available for mobile applications.

For the first method, in Concept UI based application you could read the IP by using the following code:

```
include Application.con
include RLabel.con
include GeoIP.con

class MyForm extends RForm {
    MyForm(Owner) {
        super(Owner);

        // get the remote IP
        var ip = GetRemoteIP();

        var g = new GeoIP();
        // open the GeoLiteCity database
        var arr = g.Open("GeoLiteCity.dat");
        var location = "-1, -1";

        if (arr)
            location = arr["latitude"] + ", " + arr["longitude"];

        var label_location = new RLabel(this);
        label_location.Caption = location;
        label_location.Show();
    }
}

class Main {
    Main() {
        try {
            var Application = new CApplication(new MyForm(null));
            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            echo Exception;
        }
    }
}
```

This will give you an approximate location of the user (with an error of 5 miles ore more). This method works for desktop computers and mobile applications.

The second way, works only with phone and tables (Android and iOS).

```
include Application.con
include RLabel.con

class MyForm extends RForm {
    MyForm(Owner) {
        super(Owner);

        // query the client for location
        var location = CApplication.Query("Location", false);
        // show only the last location
        if (location)
            location = StrSplit(location, ";")[0];
        var label_location = new RLabel(this);
        label_location.Caption = location;
        label_location.Show();
    }
}

class Main {
    Main() {
        try {
            var Application = new CApplication(new MyForm(null));
            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            echo Exception;
        }
    }
}
```

This method will give an exact location. Note that is no guarantee that the GPS location will be used. By default, Concept Client doesn't turn the GPS on, because it will drain the battery. It uses some passive location methods, or cell tower locations, which have almost zero impact on the battery.

Note that the Query(”Location”, false) function call may return a series of locations, separated by “;”. The format is “latitude_now, logitude_now; previous_latitude, previous_longitude”.

A solver may need to link different end-user actions with locations. For example, in a on-site support service, a dispatcher may track the site teams by using mobile concept applications.

This may also be useful for refining searches in applications or attaching coordinates to pictures. However, this system should not be used as a sole localization method for emergencies because its precision is variable. It may range from a few meters to 10 miles or more. In general, the localization is exact in cities, but rather approximate for rural areas.

You may want to overlap the location data with map services, for example, Google Maps.

MobileLocationTest.con

```
include Application.con
include RWebView.con

class MyForm extends RForm {
   MyForm(Owner) {
      super(Owner);

      // query the client for location
      var location = CApplication.Query("Location", false);
      // show only the last location
      if (location)
         location = StrSplit(location, ";")[0];

      var webview = new RWebView(this);
      var content = ReadFile("GoogleMaps.template.html");
      content = StrReplace(content, "[CENTER]", location);
      webview.Content = content;
      webview.Show();
```

```
    }
}

class Main {
    Main() {
        try {
            var Application = new CApplication(new MyForm(null));
            Application.Init();
            Application.Run();
            Application.Done();
        } catch (var Exception) {
            echo Exception;
        }
    }
}
```

The used template:
GoogleMaps.template.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta name="viewport" content="initial-scale=1.0,
            user-scalable=no" />
        <meta http-equiv="content-type" content="text/html;
            charset=UTF-8"/>
        <style type="text/css">
            html, body {
                height: 100%;
                margin: 0;
                padding: 0;
            }

            #map_canvas {
                height: 100%;
            }

            @media print {
                html, body {
                    height: auto;
                }

                #map_canvas {
                    height: 650px;
                }
            }
        </style>
        <script type="text/javascript"
            src="http://maps.googleapis.com/maps/api/js?sensor=false">
        </script>
        <script type="text/javascript">
            function initialize() {
                var myLatlng = new google.maps.LatLng([CENTER]);
                var myOptions = { zoom: 13, center: myLatlng,
                    mapTypeId: google.maps.MapTypeId.HYBRID}
                var map = new
                    google.maps.Map(document.getElementById("map_canvas"),
                    myOptions);
            }
        </script>
    </head>
    <body onload="initialize()">
        <div id="map_canvas"></div>
    </body>
</html>
```

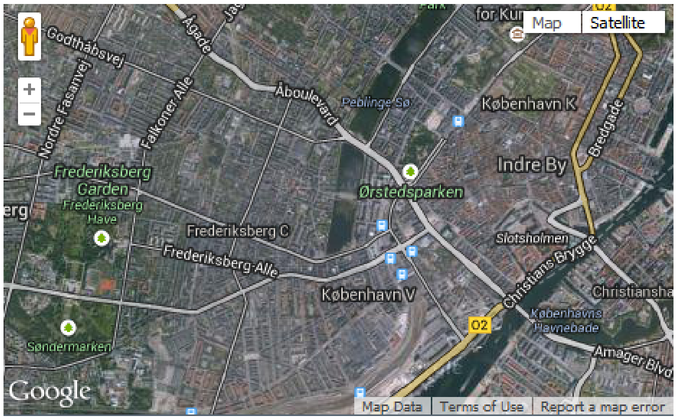


Figure 15.1: MobileLocationTest.con output

The output is shown in figure 15.1.

The localization can be integrated in a wide range of applications.

Chapter 16

GyroGears - the CAS application generator

16.1 What is GyroGears and why use it

GyroGears is an MDE tool developed for the Concept Application Server that can generate production-ready multi-user, transactional applications. In two words, GyroGears is an application generator. Model-driven engineering (MDE) is a software development methodology which focuses on creating and exploiting domain models (that is, abstract representations of the knowledge and activities that govern a particular application domain), rather than on the computing (f.e. algorithmic) concepts[1].

GyroGears automatically generates database-oriented applications for CAS (applications that run in Internet/Intranet) by just specifying the entities. Entities are somehow high level equivalents of database tables, except that an entity can be represented on more than one database table.

Gyro will also generate PDF reports, XML/CSV exporting/importing schema, automatically design the data forms, and even generate the help files for the application. The resulting application will also contain the CIDE project file, so you can modify it manually, but this should not be the case.

[1] http://en.wikipedia.org/wiki/Model-driven_engineering on January 31, 2014

Gyro is great for creating CRM, HR, EMR[2], ERP[3] or any kind of data-oriented application applications.

All Gyro applications are based on entities, actions, calendar events, reports and custom Concept sources.

An *entity* is the basic data element description. Each entity may have multiple members. A member is a data element property. For example, *User* may be an entity, having *Username* and *Password* as members. Entities may be grouped by menu category and entity's members may be organized into an unlimited number of categories and subcategories.

Members are defined as high-level data types. The available member types are:

short string
A string, up to 255 character in length

long string
An unlimited string

secret/password
A secret string, up to 255 characters in length

integer
An integer number

decimal
A decimal number

date/time
Date and/or time

file
A file

boolean
A boolean value (true or false)

choose radio
A list of radio buttons

[2] Electronic Medical Record

[3] Enterprise Resource Planning

choose combo
A combo box, containing a list of possible values

relation
A relation with another entity

non homogeneous relation
A relation with multiple entities

reference
A reference to an existing relation

picture
A normalized picture

multimedia
A normalized video and/or audio file

direct query
A direct SQL query (for SQL-based applications only)

custom function
A custom Concept function

plug-in
A custom plug-in, for example Google Maps

Each entity definition will create a Concept class, with the same name, replacing all the non-alphanumeric characters with underscore (“_”). For example, an entity named ”Client (new ones)” will have the corresponding Concept class *Client__new_ones_*. Each member, will generate a public variable in the concept class, following the same rules for non-alphanumeric characters.

File members will generate two Concept class members, one with the same member as the entity member, and one adding the *_filename* suffix. For example, if a member is called *Document*, two variables will be created, *Document* and *Document_filename*.

Picture members will have 3 additional members, adding the *_preview*, *_thumbnail* and *_filename* prefix. For example, a picture member called Image, will generate *Image*, *Image_preview*, *Image_thumnail* and *Image_filename* concept class members.

Multimedia members will have 4 additional members, adding the *_preview*, *_thumbnail*, *_totaltime*(total play time in seconds), *_status* and *_filename* prefix. The *_status* member will be "Pending", "Converted", "Error" or "Inconsistent", according to the multimedia file normalization status.

Each entity will have a *Write(Connection[...])* method causing the data to be stored in a database (SQL or NoSQL).

Actions are custom functions that the user may trigger by pressing a tool button or menu item. An action may have an associated progress bar and history. It may be used to process long-timed operations, for example, sending e-mails to multiple recipients, or creating invoices for all the clients database.

Calendar events are associations between the entities and the calendar. The calendar may be exported as iCalendar files (.ical), for importing into other software or services like Outlook or Google Mail.

The Gyro reports, are defined as XML files, similar with HTML, adding tags like <datasource>. It allows the direct query for data from the database (note that this is not supported for NoSQL databases). A report may contain tables, pie charts, line charts and/or descriptive text. The resulting report will be generated as an XSL:FO template that can easily be transformed to PDF or RTF documents.

GyroGears is extremely productive for any kind of data application. It will generate concept://-based applications, including mobile versions (for phones and tablets), and http://-based applications.

For example, you could create a CRM application in under an hour or a user generated content video streaming website in about 30 minutes. Note that HTTP-based applications are a feature for Concept Application Server, not a purpose. However, advanced caching mechanisms make the these applications run very fast, with the minimum resources used.

Note that Gyro is not a solution for everything. It targets explicitly data oriented applications. For some project, one can use Gyro only for the user interface and data modeling, the actual data processing algorithms being written manually.

GyroGears may be opened by URL on any platform:

```
concept://localhost/MyProjects/GyroGears/GyroGears.con
```

On Windows, you may use the Start menu, Concept II, Development Tools and then GyroGears.

16.2 Applications, entities and members

For our first example, a basic event management application will be created. We will have a list of persons, each person with contact data, including e-mails, and user photo. Also, a list of events, describing the event, including start and end time, and the list of invited persons. We will defined a custom action that will send e-mails to all of the invited persons. The application will have, desktop PC, HTTP and mobile interfaces.

After opening GyroGears, select "Create new solution". Gyro will prompt you to select an application template. We will use the "Empty application" template, that will create an application containing just the *User* entity. You will notice then some application properties that may be set (not mandatory).

The most important of them are:

Title
The application title (human readable text)

Description
The application description (human readable text)

Icon
Sets the image to be used as the application icon. This image will be resized by the application itself, but should be around 32 by 32 pixels.

Header
Selects a header (branding image), for the application. This image should be a relatively small image (Gyro will not resize it). I recommend the use of 64 by 64 pixels.

Decimal separator
The character to be used as a decimal separator

Thousands separator
The character to be used as a thousands separator

Date format
The date format string. For example, %d/%m/%Y will display February 3nd, 2014 as “03/02/2014”.

Date/time format
The date format string. For example, %d/%m/%Y %H:%M:%S will display February 3nd, 2014, 09:30:00 PM as “03/02/2014 21:30:00”.

Use spell checker
If set to **Yes**, a spell checker will be used (see Spell checker section)

Use spell checker
If set to **Yes**, a spell checker will be used (see Spell checker section) for all long strings.

Run in fullscreen
If set to **Yes**, the application will be standard maximized by default (recommended)

Optimize for touchscreen
If set to **Yes**, the application will use an on-screen keyboard had use slightly larger icons, for easier interaction with touch screens. Note that this refers to desktop computers having a touch screen. Mobile and tablet version are optimized for touch screens by default

Init callback function
A function to be called when the application first loads. The template is

Additional login condition
Sets the log-in condition. It may be a direct SQL condition, if preceeded by “sql:”, for example: “sql: type=’Technical’” assuming that the user entity has a member called *Type*, or a Concept condition, for example: “LoggedUser.Type==’Technical’”.

```
static function Initialize(Sender);
```

Where Sender is either a *MainForm* or a *MobileMainForm*. Note that this function is not called for HTTP applications. Note that this field must contain only the function name, without the parameters, for example *Misc::OnInitApplication.* This function may be used for general initialization, when needed, for example, checking currency rates, before the user loads the application.

User data entity

Selects the function holding the user data. This entity is used in the application login process. The user entity must have at least the following members (case sensitive): *Username* (short string), *Password* (secret), *Full name* (short string), *Level*(integer) and *Last login* (date). Additional, the members: *Last IP* (short string), *Failed attempts*(integer), *Last session*(short string), *Process ID* (integer) will provide various information about the logged user, for example, the last ip from which the user logged in, or the process id, for better identifying the user process on a server. If *Failed attempts* is defined, the user account will be automatically locked after 3 failed logins. The user entity may optionally implement a *Defaults* category, containing default values for various fields (defined in other entities). For example, if it contains a field named "Name" in the *Defaults* category, when an user creates a new object for an entity containing a member also called *Name*, the *Name* field will be automatically filled with the value in the user defaults *Name* member. The *Level* property is used for setting the entity operation rights (f.e. add, modify, delete). Additionally, some boolean members for regulating different user rights may be added, for example, a boolean member called *User/Add* will enable the user to add another user if set to true. This is effective only if the user level is below the entity add operation level as you will see later. The possible boolean members, in this case are: *User/Add, User/Modify, User/List, User/Search* and *User/Delete*. This is applicable for every entity, not just the user.

Allow anonymous login

This allows the login without user/password as a level 0 user (lowest possible level).

Generate Web 2.0 interface

If set to *Yes*, Gyro will also generate the HTTP version of the application.

Generate mobile interface

If set to *Yes*, Gyro will also generate the mobile version of the application.

Generate Web 2.0 sign up
If set to *Yes*, and *Generate Web 2.0 interface* is also set to *Yes*, it will generate a sign-up script, for the user to be able to create its own user. This is very useful for user generated content HTTP applications.

PDF Command
Sets the XSLT:FO command for processing fo files into PDF. Default the Apache FOP processor is used (must be installed on the target system).

Use cool effects
If set to *Yes*, Gyro will use operating-system dependent effects, like aero-style transparency (on Windows systems only).

Generate high level APIs
If set to *Yes*, the GyroGrease ORM APIs will be generated. GyroGrease is provides a higher API level for dealing with the GyroGears objects. This is extremely useful when writing custom code[4].

Database connection method
Specified the database connection method. The preferred method is "Native driver".

Database rules
Specified the database engine to be used. May be an SQL engine or a NoSQL engine, like MongoDB.

Datasource
Specified the data source name, when using ODBC, or the connection string. This method is not recommended, Concept providing a set of native drivers for directly connecting to the database server, without any need of ODBC.

We will only set the *Title* property to "My first Gyro Application", and will select as *Database connection method* to "Native driver", and *Database*

[4] Object-relational mapping (ORM) is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language (Source: http://en.wikipedia.org/wiki/Object-relational_mapping, on February 3rd, 2014).

rules to SQLite. Also, the *Generate mobile interface* and *Generate traditional Web 2.0 interface* will be set to *Yes*.

The entities are added by right clicking in the application tree view, and selecting add entity.

Each entity has a set of properties, the most important of them being:

Name
The entity name (may contain any character, spaces)

Public name
The entity name, as shown to the end-user. If left blank, the entity name will be used

Comments
Comments or hints shown to the user. Also, this is added to the automated help.

Render using same space as
Another entity may be selected to share the same space. For example, an entity called *Customer* may share the same space (same table or collection) with another entity called *Contact*, both of them using a few common properties, like *Name* and *E-mail*. This way, you can give different rights for different fields, by allowing some user to edit only the contact related fields for a client.

Use aditional search database
If set to *Yes*, an additional Xapian database will be used, allowing the user to perform natural searches on the data.

Use synchronous indexing
By default, the Xapian data is idenxed asynchronously, by using a background Queue, called _DoIndexForEntityName.con, where *EntityName* is replaced by the entities normalized name. If this property is set to *Yes*, the data will be indexed when is added/modified, resulting in a simpler system (no background queues), but with some latency when adding data in big sets.

Menu category
Sets the menu category for the entity. This is a free text field. Entities having the same category will be displayed in the same menu.

Read only condition

A condition making the object read-only, for example, if an entity has a boolean member called *Closed contact*, you can set the read-only condition to: *Closed_contract = true*

Archive condition

Similar with Read-only condition, but instead of opening the object as read-only, it automatically sends the object to archive.

Columns in data forms

By default, each entity is shown on a two-column form. Each member occupies a column cell or an entire row, depending on its type (for example, long text members occupy a full row).

Columns in mini forms

Sets the default form column count for embedded forms (forms shown for relational members as embedded forms).

Representative member

Selects the most important member of the entity (eg: for *Customer*, may be *Name*). This member will be shown when the object will be quick referenced.

Icon member

Selects a picture or a multimedia member to be shown in icon views (if enabled), representing the given entity.

Template member

Selects a relation member to be used as a template. Templates allow the user to automatically fill some values in the given object, from the default values stored in the template. For example, we could have an entity called *Category*, and another entity called *Product*, having a relation member called *Base category*, related to *Category.* The *Base category* member is then set as template member. Both Category and *Product* have a member called *Color.* Assuming that *Category* has an object, created Vegetable with color set to green, when the user selects “Vegetable” as *Base category*, the *Color* member will be automatically set to “Green”.

View count member

If set to an integer member, it will automatically increment the member on each object view.

Short preview expression

If set, instead of the *Representative member*, this expression may be used. For example, an entity called Customer may have two members: Name and Surname. Both of them are important when quick previewing the entity. In this case, you could set: *Name + “ ” + Surname* as the preview expression.

Mark mandatory fields

Sets the markings for mandatory fields (nefault is Bold).

Sort

Sets the sorting order. Default is ascending, but may be changed to descending.

Presentation categories

Sets the member presentation category/subcategory list. The categories are separated by comma, and the subcategories are separated by slashes from their categories. For example: *General information, Billing/Invoices, Billing/Payments* will create two categories (General information and Billing) and two subcategories for the Billing category (Invoices and Payments).

Show categories in

Sets the category UI hint. By default, categories are shown in notebooks, but that can be changed to frames or expanders.

Show subcategories in

Sets the subcategory UI hint. By default, subcategories are shown in frames, but that can be changed to notebooks or expanders. Note that not all categories may be supported on all platforms, but notebooks and frames are similar on HTTP, mobile and desktop pc applications.

Row marking method

Each object/row may have a marking hint. For example, if set to “Color”, the given row will use the color returned by the Row marking function. Alternatively, text, markup, progress bars or images may be used for marking a row.

Row marking function

Returns the object markings flags. The function form is:

```
static function Mark(obj);
```

For example, if *Mark* is defined in the *Misc* class, a correct value for this field would be *Misc::Mark*. Assuming that the target entity has a member called Value, we could set to red all the rows with a value less than 10:

```
class Misc {
    static Mark(obj) {
        if (obj.Value < 10)
            return "#FF0000";
        // empty string means default color
        return "";
    }
}
```

In a similar mode, the Mark function, may return *RImage* objects containing the needed image. As a suggestion, avoid creating too many images. Is better to create a few image that would get recycled for each call. For this, you could use the GLOBALS() function, that stores a global array (the only global variable used by Concept).

```
class Misc {
    static Mark(obj) {
        if (obj.Value < 10) {
            var img = GLOBALS()["LessThanZero"];
            if (!img) {
                // no parent
                img = new RImage(null);
                img.Filename = "res/someimage.png";
                img.Show();
                // store this image as a global variable
                GLOBALS()["LessThanZero"] = img;
            }
            return img;
        }
        return null;
    }
}
```

The use of global variables is strongly discourages, but in some limited cases, they are needed. For progress column, you just need to return the progress value as a number in the [0..100] interval.

Row hint function

Each object/row may have a hint, showed as a string. The function has the following prototype:

```
static function GetHint(Connection, obj);
```

It should return a string to be shown as a hint to the user, when it hovers the mouse over the given row/object.

Show parent in views
If set to *Yes*, a special column will be added in views, showing the object's parent. For example, Customer has a relation member called Invoices, in an exclusive relation with Invoice. When listing all the Invoices, the Customer name will be shown as a special column.

Parent level
Specifies the parent level. For example, Customer has an exclusive relation to an entity called Contract, having an exclusive relation to an entity called Invoice. If you want in an Invoice master view, to see the customer name, then parent level should be 2 (parent level 1 is contract, 2 is customer).

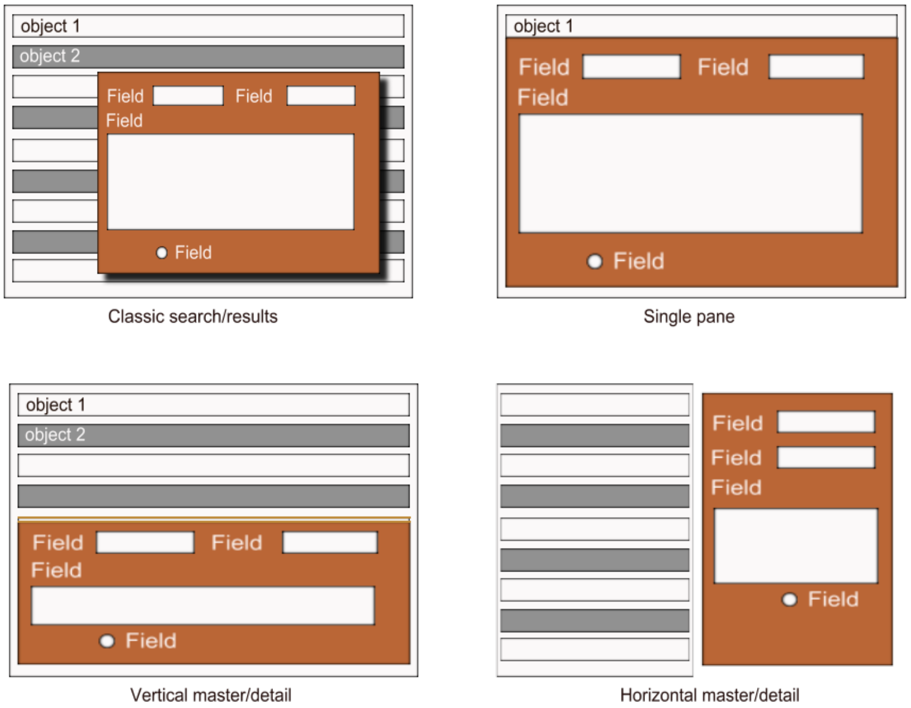
Show parent as chain
If set to *Yes* and *Parent level* is 2 or more, all the parents will be shown, each in its own column.

Search by parent
Enable the user to search by parent.

Child search level
If more than 0, when a quick database search will be performed, the system will also look in the related entities. If set to 1, just the first level of child entities will be looked up. If set to 2, it will also look into the child's children.

Master UI Model
Sets the master user interface model. It may be "Classic search/results", "Vertical master/detail", "Horizontal master/detail" or "Single pane". See figure 16.1.

Child UI Model
Sets the child view user interface model. It may be "Classic" or "Single pane". See figure 16.1.



Classic search/results

Single pane

Vertical master/detail

Horizontal master/detail

Figure 16.1: Application UI design patterns

Figure 16.1 illustrates the basic UI patterns in data-driven applications.

For each entity, the operating level and rules may be specified by using the operation and rights properties. Usually you will need to set the following properties:

Enable find
: If set to *No* the search features will be disabled for all users

Find level
: Sets the minimum user level that is allowed to search this entity (see the *Level* member for the *User* entity)

Aditional find rights
: Additional rights for users of lower level. May be a combination of same user, same group or neither. This allows a user that has no search rights, to search objects that are owned by him or her.

Enable list

If set to *Yes*, the views will automatically load some sorted objects, without any search being made.

List level

Set the minim user level that is allowed to list and browse to the objects.

Aditional list rights

Additional browsing rights for users of lower level.

Filter user by member

If an user has no listing rights, but the given entity has, for example, an *Responsible* member, a non-exclusive relation to the User entity, you may set it as a filter for user rights. This means, that if the *Responsible* references the current logged in user, it will be available for browsing.

Filter by level

Sets the minimum user level for using *Filter user by member.*

Enable add operations

If set to *Yes*, the an user will be free to create an unparented object. When set to No, these objects can be only added from exclusive relations.

Add level

Sets the minim user level allowed to add an object.

Public data

If set to yes, the data will be public on HTTP applications. This means that the data will be available as read-only to anonymous users.

Enable modify

If set to *Yes*, objects can be modified

Modify level

Sets the minimum user level allowed to modify an object.

Aditional modify rights

Additional rights for users of lower level. May be a combination of same user, same group or neither. This allows a user that has no modify rights, to modify objects that are owned by him or her.

Enable delete
Enables the delete operations

Delete level
Sets the minimum user level allowed to delete an object.

Aditional delete rights
Additional delete rights for users of lower level. For example, an user may delete only his or hers records.

Controlling the user rights, allows you to manage all the end-user operation for the Gyro application.

The following properties will control how the entity will behave in views and in relation with the end-user:

Is settings entity
If set to *Yes*, the entity will be defined as a singleton. A singleton is an entity that has only one object (avoid setting Generate views to *Yes* in this case).

Reorderable
Allows the user to reorder the objects in the master views by using the mouse and drag and drop operations. This operation is not available on HTTP application.

Autosave (disable Cancel option)
Disables the Cancel option in data forms, forcing the user to save the modified data

Generate views
If set to *Yes*, will generate master views for the entity. It will generate a list of entity's object in the main form.

Owned
If set to *Yes* two virtual members will be added to the entity: _UID (Owner user ID, the user who created the object) and _MID (ID of the user that last modified the object).

Group owned
If set to *Yes*, the _GID (Group ID) virtual member, will be added, holding the ID of the group object.

Archive

If set to *Yes* the entity will have an archive flag. Archived objects are hidden in master views.

Refresh in real time

If set to *Yes*, the master views will be refreshed automatically when the data object on the screen are modified by another user. This is available just for concept:// applications.

Require cancel confirmation

If set to *Yes*, when an user closes a form with unsaved data, a message box will be shown to the user, asking for operation confirmation

Show in toolbar

If set, will show the entity icon in the application tool bar

Show text in toolbar

If set, will show the entity name in the tool bar. For this to work, *Show in toolbar* must first be set to *Yes*.

Edit without locking

By default, the application will lock any object being edited. When this property is set to *Yes*, it will allow multiple users to edit the same object. This however is not recommended, because in may cause data inconsistencies.

Available in HTTP version

If set to *No*, the entity will not be available in the HTTP version of the application.

Static link in HTTP scripts

If set to *Yes*, the entity will be linked to all the generate scripts. This is useful for data that needs to be available on all the generated pages.

Available on mobile devices

If set to *No* the entity will not be available in mobile applications.

Define quick reports

When set to *Yes*, Gyro will automatically generate automatic reports, enabling the user to export data from master views to PDF, RTF, CSV, XML or Excel files.

Show views as icons

If set to *Yes*, will use icon views instead of tree views for master views. The icon view will only show the *Image member* and the *Representative member*.

Use single clicks in views

For editing in master views, use mouse click instead of double-click. This applies only to concept:// application for desktop PC.

Show lists as icons

Show child views as icon views instead of tree views.

Use single clicks in lists

For editing in child views, use mouse click instead of double-click.

List item expression

If set, will use the given expression, similar to *Quick preview expression*, as a markup field to be used in related lists and combo boxes.

Timestamped

Adds the _CTIME and _MODTIME virtual members, holding the creation time and respectively, the last modification time, as a string in the “YYYY-MM-DD HH:mm:ss” format.

Enable versioning/reviews

If set to *Yes*, previous version of an object will be stored in the database as revisions. This enables the end-user to see how did an object look before it was modified.

Show linked entities

If set to *Yes*, it shows all the linked entity objects. This is useful for non-exclusive relation, when the user may want to know what objects are linking to the given object.

Show as tree when possible

If set to *Yes*, it will group the similar objects in a tree view, after the given *Aggregation* expression.

Aggregation

Sets the aggregation funciton. For example, you could group objects in a master view by _UID member, to group objects belonging to the same user. In this case the expression is: “” + *_UID*. This will convert the _UID member to a string and use it as a key for aggregation.

Page size

Sets the number of items in a view page.

Entities may also have triggers. Triggers are functions that are automatically called when an entity object is modified.

The triggers are:

Validation function

This function is called just before an entity is written in the database. It returns true if the calling object is valid.

```
static function Validate(Sender, obj, var error_text);
```

Sender is the sender form (may be null), *obj* is the entity object, and *error_text* is the human readable error text to be shown to the user. This function must return true for valid objects or false if the given object is not valid.

Default click function

This function is called when the user double clicks an object, usually replacing the edit operation. The function prototype is:

```
static function EntityClick(Sender, obj);
```

Sender is the sender form, *obj* is the entity object. If this function returns -1, the default behavior will follow. If it returns 1, the application will reload all the objects (assuming the object was modified). Any other return values will be ignored.

Create trigger function

This function is automatically called after an object was successfully inserted into the database.

```
static function EntityCreate(Connection, obj);
```

Connection is the database connection handle, and *obj* is the target entity object added to the database.

Delete trigger function

Called after an object was deleted.

```
static function EntityDelete(Connection, obj_or_id);
```

Connection is the database connection handle. *obj_or_id* is the target entity object or the entity ID. You should check the type by checking *classof obj_or_id*.

Update trigger function

This function is called after an entity object was updated in the database.

```
static function EntityUpdate(Connection, obj, prev_obj);
```

Connection is the database connection handle, *obj* is the updated entity object. *prev_obj* is the previous version of the object. Note that this parameter may be null.

Archive trigger function

Called after an object was archived or unarchived.

```
static function EntityArchive(Connection, obj_id,
    archive_flag);
```

Connection is the database connection handle, *obj_id* is the updated entity object ID (not the object itself). For retrieving the object itself, a call to *Entity::FindById(Connection, obj_id)* may be needed, where Entity should be replaced with the entity's normalized name. *archive_flag* is a numeric (boolean) parameter. It is true if the object was archived, false if the object was unarchived.

Link trigger function

This is called after an entity was linked to another entity via a relation member.

```
static function EntityLink(Connection, obj, string
    rel_entity_name, string rel_member_name, rel_obj_id);
```

Connection is the database connection handle, *obj* is the linked entity object. The *rel_entity_name* parameters contains the entity to which the linking is done, as a string holding the normalized entity name. Analogously, *rel_member_name* keeps the related entity member normalized name, as a string. *rel_obj_id* keeps the ID of the related entity.

Unlink trigger function

This is called after an entity was unlinked from another entity via a relation member.

```
static function EntityUnlink(Connection, obj, string
    rel_entity_name, string rel_member_name, rel_obj_id);
```

Connection is the database connection handle, *obj* is the unlinked entity object. The *rel_entity_name* parameters contains the entity to which the unlinking is done, as a string holding the normalized entity name. Analogously, *rel_member_name* keeps the related entity member normalized name, as a string. *rel_obj_id* keeps the ID of the related entity.

Before create trigger function

This function is automatically called when an object is about to be inserted into the database.

```
static function EntityBeforeCreate(Connection, obj);
```

Connection is the database connection handle, and *obj* is the target entity object to be added to the database.

Before delete trigger function

Called just before an object is deleted.

```
static function EntityBeforeDelete(Connection, obj_or_id);
```

Connection is the database connection handle. *obj_or_id* is the target entity object or the entity ID.

Before update trigger function

This function is called before an entity object update to the database.

```
static function EntityBeforeUpdate(Connection, obj, prev_obj);
```

Connection is the database connection handle, *obj* is entity object to be updated. *prev_obj* is the previous version of the object. Note that this parameter may be null.

Before archive trigger function

Called before an object is archived or unarchived.

```
static function EntityBeforeArchive(Connection, obj_id,
    archive_flag);
```

Connection is the database connection handle, *obj_id* is the updated entity object ID (not the object itself). *archive_flag* is a numeric (boolean) parameter. It is true if the object will be archived, false if the object will be unarchived.

Before link trigger function

This is called before an entity is linked to another entity via a relation member.

```
static function EntityBeforeLink(Connection, obj, string
    rel_entity_name, string rel_member_name, rel_obj_id);
```

Connection is the database connection handle, *obj* is the linked entity object. The *rel_entity_name* parameters contains the entity to which the linking is done, as a string holding the normalized entity name. Analogously, *rel_member_name* keeps the related entity member normalized name, as a string. *rel_obj_id* keeps the ID of the related entity.

Before unlink trigger function

This is called before an entity is unlinked from another entity via a relation member.

```
static function EntityBeforeUnlink(Connection, obj, string
    rel_entity_name, string rel_member_name, rel_obj_id);
```

Connection is the database connection handle, obj is the unlinked entity object. The *rel_entity_name* parameters contains the entity to which the unlinking is done, as a string holding the normalized entity name. Analogously, *rel_member_name* keeps the related entity member normalized name, as a string. *rel_obj_id* keeps the ID of the related entity.

All the triggers must be set in Gyro without any parameters. For examples, a valid value for *Create trigger function* may be

Misc::CustomerCreateTrigger. Note the triggers are mostly notifiers, and the returned result is not used. The *Before** triggers could oppose to an operation by throwing an exception, but this is not recommended. The validation function is the only trigger that could stop a write operation gracefully, providing a human readable message.

Knowing what the entity flags and triggers are, now we can return to our first application. We defined an entity named Customer, archivable, owned, timestamped and with *Generate views* set to *Yes*. We also defined the Event entity, archivable, owned, timestamped and with generated views (similar to Customer). Now we should add members to them.

All entities have a virtual member called ID. For SQL is a numeric value, for NoSQL databases is a string value. The ID property uniquely identifies the entity object, and cannot be modified by the programmer or the end-user. An empty ID or an ID of -1 values means that the object is not yet written in the database. After a successful call to Write, the ID will be automatically set.

Each member, regarding its types, has some specific properties. A list with frequent used properties for members, regardless of the type is:

Name
The member name (may contain any character, spaces). Not that must be not the same as its entity name.

Public name
The member name, as shown to the end-user. If left blank, the member name will be used

Category
The category and subcategory, if needed. The categories are defined by its parent entity.

Comments/Description
Comments that Gyro will use in the help section.

Default value
The default value for the given member

Validation

A Perl regular expression for validating the input. This is very useful for string members that need a strict format, for example an e-mail address. An e-mail address may use

```
[A-Za-z0-9_\.\-]+@[A-Za-z0-9\_\.\-]+\.[A-Za-z0-9_\.\-]+
```

as a validation expression for ensuring that the user enters a correct e-mail address.

Init function
Sets the default function. It may be a concept function or an SQL query, returning a single row and one column. In this case, the "sql:" prefix should be added, for example: *sql: select max(val) + 1 as result from table* would be a valid initialization. Note that this method works for SQL databases only. Alternatively, you may call a Concept static function, having the prototype:

```
static function MyMemberInit(obj, default_user_data);
```

The function should return the value for the given member. *obj* is the target entity object containing the member, and *default_user_data* may be null or a key-value array containing user defaults values (the User members in the Defaults category, if any). In this case, a valid value would be: *Misc::MyMemberInit*.

Attached formula
Concept expression or SQL query to be executed every time the object is modified. It must be prefixed by the *sql:* prefix, similar to the *Init function*. The SQL query must return a single row, single column result. If not prefixed by the *sql:* prefix, the expression will be regarded as a standard Concept expression. For example: *Value * Quanity* will return the product between a member called *Value* and a member called *Quantity*.

Read-only condition
Sets the read only condition, for the given member. It is similar to the read-only condition for the entity, but applies only to the current member.

Hint provider function
Calls a function for setting the hint for the used Concept UI object rendering the member.

```
static function ProvideHint(Sender, UIControl, obj);
```

The sender is the form showing the *obj* entity object, and *UIControl* is the control used to render the object. It may be a RTextView for long strings, REdit for short string, integer or decimals, a RTreeView or RComboBox for relation and a generic box for other data types.

Mandatory
If checked, the member will be mandatory (a value must be set). For string data types, at least one character will be needed, for integer and decimal, a non-zero value will be required. Note not to check mandatory for boolean members, except for terms-and-condition style entities (where the user needs to check that has read some information), because it will force the user to set the member to true.

Unique
If checked, force the value to be unique for the entity. If the member is not mandatory, an empty value will be not checked for uniqueness. This applies to short strings, long strings, decimal, integers, booleans, choose combos and radios. For multimedia, picture and file data types, this will apply only to the movie/picture or file name. For non-exclusive relation, if set, will ensure that a related object is added just one time per member.

Reload original unique data
If checked, for unique members, will prompt the user to load the original object having the same value as the entered one.

Enforce user link
If set, will perform the uniqueness check only for entity objects created by the logged user.

Advanced search
If set, will generate the advanced search filters for the given member

Use in sorting
If set, will use this member in the default sorting

Settable to all
Generates the “Set to all” option in the concept:// application UI, enabling the end-user to set the same value for multiple objects simultaneously.

Pattern search

When using simple database search, automatically look for patterns, instead exact matches. For example, the end-user will be able to search for a fragment of the customer name, instead of forcing an exact match

Trim

For short string and long string members only, if set, it will automatically trim extra trailing and/or leading spaces or new lines from the user input.

Indexable

If set, and the *Use additional index database* is set to true for the database, this member will be indexed by Xapian. This is available only for string decimal and integer members.

Hide in forms/views/title

Hides a member in the views or forms. Also you may only hide the title, letting the field visible.

Generate quick statistic

If set, generates a quick pie chart on the concept:// application home screen, with all the distinct values for the given value. This is recommended only for fixed sets values, for example, choose combos and radios, booleans and non-exclusive relations with small lists. For bigger datasets it may generate some performance problems.

Editable in views

If set to true, the given field may be editable directly in views, without any need to open the entity form.

Read only

If set, the member will be read-only

Use alternative widget

If set, the small widgets will use the one entire form row, and the big widgets, like long string members, will use small amounts of space.

Not available on HTTP/mobile

If set to true, the member will not be available on HTTP or mobile versions

Triggers update on focus out
If set, it will trigger the attached formula update for other members. For example, a member called *Quantity* may trigger an update to a member that performs a computation, for example *Value * Quantity.*

As a special case, relation members implement some additional properties, like:

Relation with
Selects the entity to relate with

Relation type
Selects the relation type. For exclusive relations, it may be *one to one, one to at least one, one to any (zero or more)* or *one to custom.* For non-exclusive relations, it may be *many to one, many to at least one, many to any (zero or more)* or *many to custom.* The ** to one* relations accept exactly one item.

Duplicable
If this flag is set, when an user duplicates the parent entity (the one having the relation member), it will also duplicate this relation. If the relation is exclusive, it will duplicate all the child objects of this member.

Reorderable
Allows the user to reorder the items in the relation by using the mouse (not available on HTTP version)

Add by
This applies only to exclusive relations with entities having a non-exclusive relation. For example, *Invoice, Sold Product* and *Product* are 3 entities. Invoice has an exclusive relation with Sold Product, that has a non-exclusive relation to Product. Then, for Invoice, the end-user could create *Sold Product* objects, by selecting only selecting the *Product* from a drop-down list. This is called a triple relation.

Filters
This applies only to non-exclusive relations or triple relations. It allows the filtering of the available data by various members and complex conditions. All the members are separated by slash ("/").

Direct SQL filters are enclosed by [ ]. Each filter by have one or two components. The components are separated by comma. For filters related to other members of the same entity, just one component is used. For complex filters, two components are needed, the first one indicating the source date, and the second one indicating the path to the current entity. For example, assuming that we have Customer that has a relation member called Contracts. Each Contract has multiple Subscriptions and each Subscription has multiple Services related to Service types (a service template). Assuming that we have a ticketing application, and an end-user wants to open a ticket for a client, only for the active services from any of the customer's contracts, the filter string would be:

```
/Customer/Contracts[suspended=0]/Subscription/Services/Service_type,
    /Customer/Tickets
```

Note that only the root element uses the entity name (in our case, Customer). The path elements use only member names. This is implemented this way because an entity may have multiple relation with the same entity, by using different relation members. The "[suspended=0]" is a direct SQL condition, that will be evaluated by the database server directly, for the corresponding query. This is only available for SQL-based database engines. This is the Gyro equivalent of a join query. This will show only the service type on which the customer has an open subscription. The second path, simply represents the path from the *Customer* to our opening *Ticket*, the Customer being the common parent for the *Service* and for the *Ticket*. After stepping over the next few sections you will better understand this.

Accept multiple selection

For non-exclusive relations, enables the end-user to select multiple objects at once, by keeping the shift key pressed.

Enable 'send to option'

Enable the "Send to" (for non-exclusive relations) or "New for" (for exclusive relations), in the related entity master view, enabling the user to add items th the current object without opening it first.

Use embedded window

When editing an entity, instead of using another window will render the related form embedded in the entity form.

Use a combobox
Only for *many to one* relations, use a combo box instead of a tree view for rendering the relation member.

Use editable box
For relations rendered as combo box, use an editable combo box, allowing the user to search items in the combo.

Strict/one per user
Allow just one child item per user.

Returning to the first Gyro application, containing the Customer, Event and User entities, we should start adding the members. A member is added by right clicking the parent entity, and selecting “Add member”.

For *Customer*, add the following members:

Name short string, mandatory, trim, pattern search and advanced search set

E-mail short string, unique, mandatory, trim, pattern search and advanced search set. The validation will use the validation (Gyro should auto suggest it):

```
[A-Za-z0-9_\.\-]+@[A-Za-z0-9\_\.\-]+\.[A-Za-z0-9_\.\-]+
```

Notes long string, trim, pattern search and advanced search set

The *Event* entity will have the following members:

Start date with attached time, mandatory, advanced search

End date with attached time, mandatory, advanced search

Description long string, trim, pattern search and advanced search set

Invited customers relation with Customer, many to any (zero or more)

The *Event* entity will also be visible on the calendar. For this, a calendar event should be created, by right clicking on application tree view and selecting “Add calendar event”.

Each calendar event should have an associated entity, in our case, *Event*, a start member, *Start*, an optional end member, *End*, and an event description member, *Description*. This event will be called *Scheduled event*.

By right clicking on the application tree view, and selecting “Add action” we will defined a custom action, called *Notify clients*. The trigger function will be *Misc::SendEmails*. The action function prototype is:

```
static boolean MyAction(Sender, ProgressForm, parameter=null);
```

Where *Sender* is the MainForm and ProgressForm is a custom form, displaying the progress. If the functions returns non-zero, the progress form will not be closed automatically, allowing the end-user to read all the output.

16.3 High level APIs

Gyro provides two level of high level APIs: the Gyro standard APIs and the GyroGrease optional APIs. The standard Gyro APIs are automatically generated based on the entity description.

A list describing all the entity APIs is provided bellow. The common parameters are:

Connection is the database connection object

no_blobs is a boolean flag. When is set, the returned object or objects will not contain large fields (for example, files or pictures) and will be limited to the short preview length.

UID the logged user ID. When is set to -1, this parameter will be ignored. This parameter will usually be available just for entities having the *Owned* property set to *Yes*.

GID the logged user group. ID When is set to -1, this parameter will be ignored. This parameter will usually be available just for entities having the *Grouped* property set to *Yes*.

Each entity has at least the following data members:

ID read only property, keeping the object id. If ID is -1, the object is not yet written into the database. A call to *Write* will then set the ID to the auto-generated object id.

__USERDATA a free variable member, to be used by the programmer (not used by the framework).

__CHANGED_FLAG = false meta-data field, used by the UI. If a value of true means that the object has unwritten changes.

__VIEWINDEX = 1 meta-data field, keeping the object index in a set, when a multiple object result is returned

The APIs are generated individually, for each entity. They may vary in parameters, according to the entity's settings. The main APIs are:

object FindById (Connection, id, no_blobs=false, lock=false[, is_revision_id=false], UID=-1)
id is the id of the object. If *lock* is set to true, a write lock will be held on the object. The lock will be released when the current transaction ends. When an entity has the *Enable versioning/reviews* property set to *Yes*, it will use two actual tables, one for database objects and another one for the previous versions of the same object. In this case, *is_revision_id* parameter will be present, and if set to true, the *id* look-up will be in the revision table. The function will return the entity object with the given *id*, or null if not found.

_Clone (keep_dbid=true, clone=null, number clone_metadata=true)
Will clone the current object and return a new object, identical with the original. If *keep_dibid* is set to true, the cloned object will have the same id with the original. If the id is not kept, a call to *Write* will cause an insert operation in the database. *clone* may contain an object to be used (instead of creating a new one). *clone_metadata* will cause the cloning of all entity meta data (f.e. __USERDATA).

_CloneRelations (Connection, source_id, UID=-1, GID=-1)
Copies all the relations used by the object from the given source object id. For exclusive relations, object may be copied, depending of their *Duplicable* flag settings. If UID and/or GID is set, only the objects owned by the given user id and/or group id will be copied.

Write (Connection[, UID=-1][, GID=-1], prev_object=null)
Will write (insert or update) the object in the database. The *UID* parameter will be only present for entities having the *Owned* property set to *Yes*. GID will be present for entities with *Grouped* set to *Yes*. prev_object is an optional parameter containing the original object (before modification)

Delete (Connection, children=true)
Deletes the current object from the database. If children is true, it will delete in cascade all its children (based on exclusive relation members).

static DeleteById (Connection, id)
The static version of *Delete*, using the *id* instead of the object itself. This is a convenient way of deleting when having only the id, avoiding an extra call to *FindById*.

static Archive (Connection, id, arc=true, children=true)
Archives (when *arc* is true) or unarchives (if *arc* is false) the object identified by *id*. If the *children* is set to true, it will also archive or unarchive, in cascade, all the children, based on the exclusive relation members.

array _Format (array objects, mime_encode_blobs=false)
Formats the current objects, for displaying to a human. *objects* is an array containing the objects to format. If *mime_encode_blobs* is set to true, all the encountered blobs will be encoded in base 64. The function returns the *objects* array received as parameter.

_GetParent (Connection, no_blobs=false)
Returns the object's parent (related via an exclusive relation), if any. If the object has no parent, the function returns null.

NormalizeMultimedia ()
This is defined only for entities that have at least one multimedia or picture field. This function will convert the multimedia or picture members to the normalized values defined in Gyro interface.

static array ISearch (Connection, var estimated, string what, start=0, len=(entity page size), sort_by="", descending=true, no_blobs=false, arc=false, UID=-1, GID=-1)
This function is defined only for entities using additional search database. It performs a probabilistic search for the given query (*what*). The *estimated* parameters is set to the estimated object count matching *what* across the entire database. *start* is the offset of the object and *len* the number of objects wanted. *sort_by* is the normalized member name to sort upon. If descending is set to true, the sorting will be done from the highest relevance to the lowest. If *arc* is set to true, the search will be done on the archive instead. *UID* and *GID* may filter the results for objects owned by the given user or group. This function returns an array of objects matching the given search.

static number GetCount (Connection, string criteria[,arc=false], extra_fields=null, UID=-1, extra_direct_query="", extra_where=""[, GID=-1])
Returns the number of objects in the database matching the given *criteria*. The *arc* parameter is present only for entities having the *Archive* flag set to *Yes*. The *extra_fields* may be a vector describing additional filters, having arrays as elements. For example, if the entity has a member called "Value" and we want to look for objects with a value between 2 an 10, we can use an filter like [["value", ">=", 2], ["value", "<=", 10]]. For relation members, a filter must have 5 elements, containing ["target_entity_name", "parent_entity_name", "member_name", operand, array ids], where operand is either "=" or "<>" and ids is an array containing all the target object ids. This is a convenient way of filtering data, abstracting the database layer (works for both SQL and NoSQL). The extra_direct_query is reserved for adding JOIN statements and should not be avoided, being SQL syntax dependent. In *extra_where* you could specify additional where conditions to be appended to the internally generated query. Note that this works only for SQL databases.

static array GetSmall (Connection, string criteria, start=0, len=(entity page size), sort_by="", descending=true[,arc=false], extra_fields=null, UID=-1, no_blobs=true, extra_direct_query="", extra_where=""[, GID=-1])

Returns an array of objects matching the given criteria. *start* is the offset of the object and *len* the number of objects wanted. See explanation of *GetCount* for the other parameters.

string CompareTo (Entity otherobject)
Compares the current object with *otherobject*. If the objects are identical, an empty string is returned. Otherwise, a string containing all the different members names separated by new line is returned.

boolean DoValidate (Lang S, Connection, var error=null, Sender=null, ignore_fields_array=null)
Returns true if the *object* validates all the restrictions imposed by Gyro. *S* is a Lang object for translating the error messages. *Sender* may be the form rendering the object, if any. If the object is invalid, the *error* parameter will be set to a human-readable string describing the validation error. *ignore_fields_array* may be a key-value array containing all the normalized member names to be excepted from validation, as keys, with true as values. For example [“Value”: true, “Price”: true].

SetByMemberName (Connection, string membername, var newvalue)
Sets a member value by its normalized name, and returns the old value.

AsText (short_version=false)
Returns the human readable text representation of the object, to be used in reports or lists. If *short_version* is set to true, only the representative member or *Short preview expression* is used.

Note that Gyro defines a lot more APIs than the previous list. However, I do not recommend the use of undocumented API, because these are usually subject to changes, and you may need to modify your code when upgrading Gyro.

The relation members generate additional APIs. The naming convention for relations is *$NormalizedEntityName$NormalizedRelationMemberName*. For example, assuming that we have an entity named *Client* and another one called *Invoice*, and *Customer* has a relation member called *Issued invoices* defining an exclusive relation with *Invoice*, the API naming base will be *CustomerIssued_invoices*. We will refer this as *NamingBase*. When referring the relation member, the *target* entity is the entity which the member relates to. The *parent* entity is the one containing the member.

Relation member generate the following additional APIs in the *target* entity:

boolean LinkTo*NamingBase* (Connection, obj_id)

Links to current object to the given *obj_id* object id. In our example, this function will be defined by the *Invoice* entity, and named *LinkToCustomerIssued_invoices*. This function returns true if succeeded, false if it fails. It will fail if the object referred by *obj_id* is not yet written in the database. Note that this function must never be called before calling *Write*. A non exclusive relation may have multiple links, by multiple calls to LinkTo*NamingBase*, while exclusive relations are limited to just one call, else data inconsistency will result. The function itself will not check if the relation is exclusive and *target* is already linked with another object, this being up to the programmer.

Eg.:

```
[..]
var invoice = new Invoice();
invoice.Description = "Database analysis";
invoice.Quantity = 20;
invoice.Unit = "Hours";
invoice.Price_per_unit = 80;
invoice.Write(Connection);

// assuming that customer is an object of Customer
invoice.LinkToCustomerIssued_invoices(Connection, customer.ID);
[..]
```

boolean UnLinkTo*NamingBase* (Connection, obj_id)

This function removes the link between a relation member object (*obj_id*) and a target. It is the opposite of LinkTo*NamingBase*. Returns true if succeeded, false if it fails. Note that this function simply breaks a link, without deleting any objects. The *Delete* function will automatically break all object links before deleting the object. If you want to delete an object, a call to this function may be not necessary.

static array GetBy*NamingBase* (Connection, obj_id, sort_field=“”, descending=false, no_blobs=false, start=0, len=(entity page size)[, UID=-1], extra_condition=“”[, GID=-1][, is_revision=false])

This function returns an array of target objects linked to the give *obj_id* object id. Note that for relation restricted to just one object (many to one, one to one), the len parameter will default to 1. The UID parameter is present just for *Owned* targets. The GID parameter is present just for *Grouped* targets and the *is_revision* flag will be present just for targets having the *Enable versioning/reviews* property set to *Yes*. *extra_condition* applies just to SQL databases, and it holds a direct SQL condition to be appended to the generated SELECT...WHERE statement. This should be avoided because is database-engine dependent. In the given example, a list of all the invoices for a specific customer could be obtained by using the following code:

```
[..]
do {
    // get invoices is blocks of up to 50
    var invoices =
        Invoice::GetByCustomerIssued_invoices(Connection,
        customer.ID, "", false, false, start, 50);
    var len = length invoices;
    for (var i = 0; i < len; i++) {
        var invoice = invoices[i];
        if (invoice)
            DoSomethingWith(invoice);
    }
    start += 50;
} while (invoices);
[..]
```

static number GetBy*NamingBase*Count (Connection, obj_id[, UID=-1], extra_condition=“”[, GID=-1][, is_revision=false])
Returns the number of target entities linked to the *obj_id* object id. The use of this function is encouraged just for reporting purposes. When full data retrieval is needed, is better just to call GetBy*NamingBase* like in the previous example, until no more data is available.

Alternatively, the same relation generates the following members in the *parent* entity:

static array GetBy*RelatedMemberName* (Connection, target_id,

sort_field=“”, descending=false, start=0, len=(entity page size), extra_condition=“”)
Returns an array of *parent* objects linking to the *target_id* target id.

Assuming that each invoice has a non-exclusive link with an entity called *Product*, via a relation member also called *Product*, we could retrieve all invoices containing a given product by using the following code:

```
[..]
do {
    // get invoices is blocks of up to 50
    var invoices = Invoice::GetByProduct(Connection,
        product.ID, "", false, start, 50);
    var len = length invoices;
    for (var i = 0; i < len; i++) {
        var invoice = invoices[i];
        if (invoice)
            DoSomethingWith(invoice);
    }
    start += 50;
} while (invoices);
[..]
```

static number GetBy*RelatedMemberName*Count (Connection, target_id, extra_condition=“”)
Returns the number of objects linked with the given *target_id* target id. The use of this function is encouraged just for reporting purposes.

This parent-oriented methods should be rarely needed, and usually, just for non-exclusive relations.

A file called Utils.con contains various APIs for direct database queries or time and string conversions. The most used functions are:

```
class Utils {
    // executes the query on the database Connection. Returns true
        if suceeded
    static boolean DirectNonQuery(Connection, string query,
        throw_err=true);
    // executes the query on the database Connection,
    // returning either the field called result, or the first
    // field in the first row returned by the query.
    static string DirectQuery(Connection, string query,
        throw_err=true, string result_default="-");
    // limits a UTF8 string to a maximum of max_len characters.
    // If the string is cut, the after string si appended.
    static string Limit(string utf8string, max_len=50, after="..");
    // same as Limit, but at byte-level (ignoring UTF-8 characters)
    static string ByteLimit(string s, max_len=50, after="..");
    // returns the current date as a string
    static string DateNow();
    // converts t to its string representation (YYYY-MM-DD).
    // If has_time is set to true, the returned string will also
    // contain the time
    static string ToDate(number t, has_time=false);
    // extracts year, month and day from t (as returned by time()).
    // If t is 0, the current date will be used
    static GetDate(var year, var month, var day, number t=0);
    // validates a date string, and returns true
    // if the date is valid
    static boolean ValidateDate(var string date,
        date_null_val="0001-01-01");
    // validates a date and time string, and returns
    // true if the date is valid
    static boolean ValidateDateTime(var string date,
        date_null_val="0001-01-01 00:00:00");
    // escapes a string, escaping all the XML special characters
    static string XMLSafe(string utf8string);
    // copies a filed from srcfilename to dstfilename
    static boolean CopyFile(string srcfilename, string dstfilename);
    // fid may be a string or a File object. It will be written
    // to filename_and_path, creating directories if needed
    // Returns true if succeeded.
    static boolean SafePathWrite(fid, string filename_and_path);
    // creates a directory, and subdirectories recursively,
    // if needed
    static boolean DoDirectory(string path);
    // parses a string to a date. If as_array is true
    // the result will be an array, as returned by
    // localtime. If is false, it will return
    // the date as time since epoch (as returned by time())
    static number/array StrToTime(string date, as_array=false);
}
```

For a complete list, check the generated *Utils.con* file, found in the include directory of the gyro generated application.

The previous APIs don't validate the user access rights, this being entirely up to the programmer. The used objects are the one used internally by the Gyro application. However, in some cases, a higher level of API is needed. When setting *Generate high level APIs* to *Yes*, the GyroGrease APIs will be generated. The *GyroGrease.con* file must be included in order to have access to the GyroGrease APIs.

GyroGrease implements the following static functions:

***Connection GyroGrease::Connect** ()
Connects to the database using the settings in the application .ini file and returns the connection handle

GyroGrease::Begin ()
Initializes a transaction

GyroGrease::Savepoint (string savepointname)
Creates a save point with the give name

GyroGrease::Rollback (string savepointname="")
Rolls back all the database writes since a successful call to *GyroGrese::Begin()*, ending the current transaction if *savepoint* is empty. If a *savepointname* is provided, the data will be restored as it was the save point was created, without ending the current transaction.

GyroGrease::Commit ()
Commits the current transaction to the database, ending the transaction.

The most important function defined by GyroGrease is actually a macro called **Grease**(object). This function will return a new GyroGrease-aware object with overloaded members for easy access. A call to an already greased object, will return the same object. Note that GyroGrease performs checks for user rights for given operations at API level.

The GyroGrease object will have the following members:

Parent : read-only property (object)
Gets the object's parent

Lock ()

Locks the current object for writes from another connection. The lock will be released when the current transaction ends.

Delete ()

Deletes the object

Archive ()

Archives the object (if the object is archivable and the logged user has the rights)

Write ()

Writes the object to the database, if the logged user has rights for this operation.

static array Array (start=0, len=-1)

Returns a list containing the objects in the database, starting from *start* and containing up to *len* elements. If *len* is -1, all the elements will be retrieved.

For each relation member in the object, GyroGrease will create a special property class object, for managing access to the related objects. The member object will be accessed simply by its normalized name, and will have the following members:

Sort : string property

Sets the sorting member name

Descending : boolean property

If set to true, the sorting will be done in descending order

BlobMode : boolean property

When is set to true, the returned object or objects will not contain large fields (for example, files or pictures) and will be limited to the short preview length.

Count : number read-only property

Gets the related objects count

array Array (start=0, len=-1)

Returns the related objects as an array.

operator [](index_or_key)
Returns the related object on the given index, if *index_or_key* is a number. If *index_or_key* is a string, and the related entity has a quick search member that is unique, it will return the object identified by the given key (the quick search member value equal with the given key). If no unique quick search member is set, and *index_or_key* is a string, an exception will be thrown.

operator =(object)
This is defined only for relations limited to one object (many to one and one to one). Will link the given *object* with the current relation (unlinking a previous linked object if needed). If *object* is null, the previous object (if any) will be unlinked.

Add (object)
This is defined only for relations not limited to one to a single object. Will link the given *object* with the current relation.

Remove (object)
This is defined only for relations not limited to one to a single object. Will unlink the given *object* from the current relation.

Note that the GyroGrease object will always inherit all the GyroGears APIs (except the overwritten ones, for example *Write*).

Assuming that the structure in figure 16.2 is defined, the following code could be used (note that BlogExample.con must be placed in the Blog application root):

BlogExmaple.con

```
include include/GyroGrease.con

class Main {
   Main() {
      try {
         var connection = GyroGrease::DoConnect();
         GyroGrease::Begin();
            // you could combine GyroGrease with
            // GyroGears APIs
            // var post=Grease(Post::FindById(connection, 2));
            // will look for the post with 2 as an id
            post = Grease(new Post());
```

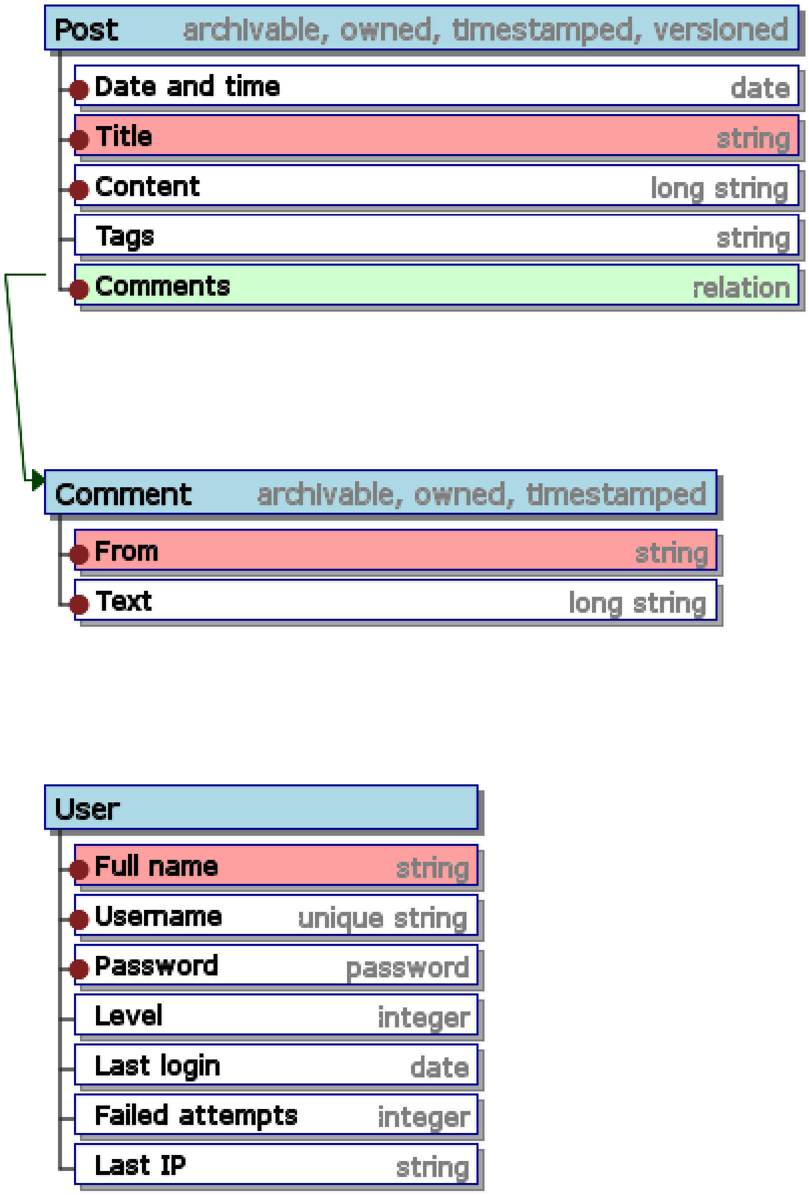


Figure 16.2: A simple blog model

```
                post.Title = "test";
                post.Content = "Hello world!";
                post.Write();

                // no other modify operation is allowed
                post.Lock();

                GyroGrease::Savepoint("there");

                var comment = Grease(new Comment());
                comment.From = "GyroGrease";
                comment.Text = "Hello world";
                comment.Write();

                // cancel the comment write
                GyroGrease::Rollback("there");

                comment = Grease(new Comment());
                comment.From = "GyroGrease again";
                comment.Text = "Hello world !!";
                comment.Write();

                post.Comments.Add(comment);

                var len = post.Comments.Count;
                echo "Comment count:" + len + "\n";
                for (var i=0; i<len; i++) {
                    comment=post.Comments[i];
                    echo "From: " + comment.From + "\n";
                    echo "Comment: " + comment.Text + "\n";
                    echo comment.Parent.Comments[0].Text;
                    post.Comments.Remove(comment);
                }
                // send the post to archive
                post.Archive();
            GyroGrease::Commit();
        } catch (var exc) {
            echo exc;
        }
        if (Connection)
            Connection.Close();
    }
}
```

Note that transactions are not supported for NoSQL database servers like MongoDB. The *Begin*, *Savepoint*, *Rollback* and *Commit* have no effect on these engines.

16.4 High-level data types

GyroGears applications are using several high level data types. Each entity member must be of one of those types.

As described before, all members, regardless of their type, have some common properties.

A quick recap of the member properties:

Property	**Description**
Name	The member's name
Public name	An alternate name to be shown to the user
Category	The category containing the member
Default value	A default value/initial value of the member
Validation	A validation PERL-compatible regular expression
Init function	A function to be called when the object is initialized. It may be a concept function, or a direct SQL query, if prefixed with "sql:", for example: *sql: select max(val)+1 from sometable*
Attached formula	A concept expression or an SQL query (prefixed by "sql:"), This expression can refer existing members, for example, you can perform an addition between a member called Price and Tax by using: *Price + Tax*

And the member flags:

Flag	**Description**
Mandatory	The member value is mandatory
Unique	Force the member value to be unique across the database
Enforce user link	If *unique* flag is set and the entity is owned, the unique check will be done only for objects belonging to the current logged user

Advanced search	Generate advanced search filters in Gyro application UI
Read only	Makes the member value read-only
Trim	Automatically trims additional spaces/tabs/new lines from the member value
Editable in views	Makes the member editable in views, without the user opening a form
Triggers update on focuse out	Recompute all *attached formulas* for all members in the entity when the member is not edited anymore
Hide in views	Hide the member in views
Hide in forms	Hide the member field in data forms
Hide title in forms	Hide the member title in data forms
Visible in PDF	Shows the member in automatically generated printed forms
Visible in report	Shows the member in quick reports (automatically generated reports)
Pattern search	Search members by fragments of their content instead of exact match
Indexable	If the entity has an additional index database, this member will be indexed
Reload original unique data	When the unique check fails, the entity having the same value is automatically loaded into the form

16.4.1 Short string

Short string are used for storing strings up to 255 characters. It has the following specific properties:

has suggested values
Generates suggestions in the data field, when the user types a value. Suggestions are based on previously entered values.

Short preview length
Sets the maximum number of UTF8 characters to be shown in views and reports.

Modifier function

Sets a normalization function. The function template is:

```
string ModifierFunction(string parameter);
```

By default, Gyro provides 6 modifiers: *ToUpper, ToLower, UTF8ToUpper, UTF8ToLower, Utils::SentenceCase* and *Utils::NameCase.*

16.4.2 Long string

Long string are used for storing strings with unlimited[5] number of characters.

It has the following specific properties:

Render as plain text

Render the long string as plain text

Render as WYSIWYG

Render as WYSIWYG[6]. The string is stored as HTML + CSS and a rich editor is provided to the user, allowing formating of the text.

Render as a sheet

Render the long string as spread sheet. This is supported only by concept:// applications running on PCs, and is better to be avoided due to limited support.

Short preview length

Sets the maximum number of UTF8 characters to be shown in views and reports.

Modifier function

Sets a normalization function. Similar to short strings, Gyro provides 6 modifiers: *ToUpper, ToLower, UTF8ToUpper, UTF8ToLower, Utils::SentenceCase* and *Utils::NameCase.*

Images source path

[5] Limited only by database server capabilities, usually $2^{31} characters$

[6] What You See Is What You Get

When in WYSIWYG mode, set the path to a relation with an entity with a image member, to be used as source for pictures. See the *Relation* data type for more information about paths.

16.4.3 Secret field

Secret field are used for storing strings up to 255 characters. Unlike *short strings*, the character will be masked (not visible to the user). This is great for storing passwords. If the **Requires confirmation** property is set, the user will be presented with two masked fields, in order to confirm the input.

16.4.4 Integer

This data type can hold signed integer values. Depending on the database server used, it can range from 2^{32} to 2^{53} without loosing precision.

16.4.5 Decimal

This data type can hold signed decimal values. Depending on the database server used, it can have values up to $1.8x10^{308}$ (the limit of the Concept number).

It has the following specific properties:

Decimals
The number of decimals

Render as chart bar
In views, instead of printing the actual value, a progress bar will be used (for values [0.00 .. 100.00]).

Show sum in reports
In reports and master views, when at least one filter is used or a search is performed, it will compute the sum for the given elements.

16.4.6 Date/time

Date/time holds a date (with or without time), as a string, in the form YYYY-MM-DD or YYYY-MM-DD HH:mm:ss, if time is needed. The null date is “0000-00-00” or “0001-01-01”.

The **Default date** property can be set to today, for today’s date, or any of the preset values (for example *tomorrow* or *a year ago*). It can also hold static date values, for example 2014-02-15 for February 2nd, 2014.

If the *Attach time* is set, the member will also hold a time value, appended to the date value.

16.4.7 File

A file member will hold a file of any given size[7]. The user will be able to attach a file to the data form.

It has the following specific properties:

Download the file when clicked
When the user will click on the file name on the data form, the file will be downloaded

Open the file when clicked
When the user will click on the file name on the data form, the file will be downloaded and the opened by the client with the default file handler (if trusted).

Store on disk instead of the database
When set, Gyro will keep the file on the disk instead of using BLOB fields in the database. This is a convenient way of optimizing the database size. Is recommended to set this property, especially when dealing with large files.

[7] Restricted only by the database server or file system

16.4.8 Boolean

A number value that can be either **true** or **false**. This will be rendered as a check box. As a note, avoid setting the *Unique* flag, because the user will be able to insert just two records in the database (one with a value of true and another one as false). Also, setting the *Mandatory* flag, will force the user to set the member to true. The *Default value* property must be set to 1, if for true or 0 for false (if not set, 0 is assumed).

16.4.9 Choose radio

This will present a list of options to the user, rendered as radio buttons.

Values
The values, separated by comma. For example: *One, Two, Three*

Allow multi-select when searching
When using advanced filters (*Advanced search* is set), the user will be able to search for multiple values at once (for example, searching simultaneously for “One” and “Two”).

Note that *Default value* must be set to a value contained by the *Values* list, for example, *One*.

16.4.10 Choose combo

This will present a list of options to the user, rendered as combo box.

Values
The values, separated by comma. For example: *One, Two, Three*

Allow multi-select when searching
When using advanced filters (*Advanced search* is set), the user will be able to search for multiple values at once (for example, searching simultaneously for “One” and “Two”).

Similar to *choose radio*, *Default value* must be set to a value contained by the *Values* list, for example, *One*.

16.4.11 Picture

A picture member will hold a normalized image. The user will be able to attach a image to the data form, by using a local picture file, a camera for capture or a copy/paste operation.

It has the following specific properties:

Resize
If set, resizes the original image to the given size (width x height)

Thumb size
Sets the thumbnail image size to be used in views

Preview size
Sets the preview image size to be used in forms

Format
The image normalization format (JPEG or PNG)

Allow webcam capture
Allows the user to capture an image from the camera

Autocrop thumbnail
Crops the thumbnail, in order to have the exact thumb size. If not set, the picture will be shrunk to fit a rectangle defined by the thumb size, maintaining the aspect ratio.

Autocrop preview
Crops the preview, in order to have the exact preview size. If not set, the picture will be shrunk to fit a rectangle defined by the preview size, maintaining the aspect ratio.

Autocrop original image
Crops the image, in order to have the exact given size. If not set, the picture will be shrunk to fit a rectangle defined by the resize property, maintaining the aspect ratio.

16.4.12 Multimedia

A picture member will hold a normalized video. The user will be able to attach a multimedia file to the data form, by using a local file.

It has the following specific properties:

Normalization video size
Sets the normalization video size. Any video file uploaded by the user will be converted to this size.

Thumbnail size
Sets the thumbnail frame size.

Preview size
Sets the video preview frame size.

Maintain aspect ratio
If set, it will maintain the aspect ratio of the given video

File format
Sets the thumbnail and preview image format (JPEG or PNG)

Compare at most (thumbs)
Sets the number of frames to be extracted and compared. The application will automatically choose the frame containing the most colors, to avoid identify black frames. Also, the system uses some random coefficients, in order to avoid for the user to hack the video, forcing a specific frame.

Normalized video format
Sets the video format to be used internaly (avi, flv, ts, mpeg, h264).

Video bitrate
Sets the video bitrate in bits

Audio samplerate
Sets the audio sample rate in hz

Autocrop thumbnail
If set, the thumbnail will be automatically cropped to have the given size.

Store on disk instead of the database
When set, Gyro will keep the media file on the disk instead of using BLOB fields in the database. This is a convenient way of optimizing the database size. Is recommended to set this property, especially when dealing with large media files.

Note that for every multimedia member, Gyro will generate a special loop script, called *ConversionService_$EntityName_$MemberName.con* located in the application root. This script must be set up as a Concept service or simply run from the console for processing the videos.

16.4.13 Custom function

A custom function that the user may call by pressing a button.

It has the following specific properties:

Function

The function to be called (without the parameters list). A valid value will be: *Misc::CustomFunction*.

```
static CustomFunction(RForm Sender, object target);
```

Sender may be the entity form, parent object form or main form (MainForm or MobileMainForm). You can access the database connection by reading *Sender.GetDBLink*() and the localization object (Lang class) by reading *Sender.GetLanguage*(). *target* is the calling object. The returning value of this function is ignored.

Called by pressing a button

If not set, the function will be automatically called when the user opens the entity form. If set, the user will be presented with a button for calling the function.

View in context menu

If set, the user will be able to call the function by right-clicking the object in views, and selecting the given member name.

Icon

Set an icon for the given action. It is recommended to use icons between 16x16 to 32x32 pixels.

16.4.14 Direct query

Performs a direct SQL query and shows the result as a read-only text. This is only supported for SQL databases, and is not recommended (it exists only for compatibility with GyroGears 1.0). The use of *Attached formula* is recommended instead.

The **Formula** property sets the SQL query to be used. The query must return exactly one row containing one column, for example:

select max(value)+1 from table;

16.4.15 Relation

Relation members are member referencing other entities. There are two kinds of relations.

In an exclusive relation, a child object can be linked just with one parent object. This child object is created within the relation and cannot be re-used in another exclusive relation.

Exclusive relations are:

Relation type	**Description**
one to one	Exclusive relation with exactly one other object
one to at least one	Exclusive relation with more than one other object
one to any	Exclusive relation with zero or more other object
one to custom	Exclusive relation with a *minimum* and *maximum* of objects

In a nonexclusive relation, a child object may exist independent of the relation, and may be linked to multiple non-exclusive relations.

Nonexclusive relations are:

Relation type	**Description**
many to one	Nonexclusive relation with exactly one other object
many to at least one	Nonexclusive relation with more than one other object
many to any	Nonexclusive relation with zero or more other object
many to custom	Nonexclusive relation with a *minimum* and *maximum* of objects

It has the following specific properties:

Relation type
Set the type of the relation

Relation with
Set the child entity for the related member. This property is mandatory.

Max items (custom relation)
When *relation type* is *one to custom* or *many to custom*, sets the maximum number of child objects.

Min items (custom relation)
When *relation type* is *one to custom* or *many to custom*, sets the minimum number of child objects.

Add by
For exclusive relations only, when the *Relation with* entity has at least a relation member itself, can be set as an *Add by* member. This will cause the user to create the new object, by creating or selecting the entity object referred by the *Relation with* relation member.

For example, an entity Customer, has a relation with Invoice, and Invoice has a relation with Product. If in the Customer/Invoices we set *Add By* Product, when the user will add an Invoice, it will be first prompted to select a product, prior creating the invoice. This is called a triple relation.

Filters
For nonexclusive or triple relations, a filter path may be provided. This is the equivalent of a SQL join.

A filter may have one or two components, separated by comma. A single-component filter will refer a member of the current entity. Assuming the following structures:

Customer: Offers(exclusive), Invoices(exclusive) Offer: Products (non-exclusive) Invoice: Product (non-exclusive)

For Customer/Invoices we have the *Add by* property set to Product, but we want to select only the products in the client offers. That can be achieved by using the filter (on the Invoices relation):

```
Offers/Products
```

This will limit the user's product option to the ones linked to the client offers. Entities and members are separated by "/" in paths. This will work both on SQL and NoSQL databases.

Additionally, for SQL databases, filters may be used. A filter is a direct SQL condition enclosed by [ ], for example:

```
Offers[expired=0]/Products[price>10.00]
```

This will present the user only with products with a price bigger than 10.00 and linked to non-expired offers.

Note that on single component filters, we only work with member names, starting from the current parent entity.

Double component filters provide more complex filters, when the relation involves a relation with an entity having a common parent with the current entity. The first component will describe the path to the target entity, while the second one will describe the relation with the current entity.

In our example, the following filter could be set on *Invoice*, instead of *Customer*, resulting in the same behavior.

```
/Client/Offers[expired=0]/Products[price>10.00],/Client/Invoices
```

Note that in this case, the first element of each filter is an entity instead of a member. It this case, the entity name is preceded by a "/".

Shortcut key

Sets a short cut key for the relation. Modifiers like <alt> may be used, for example **F3** or **<alt>F3**. Note that F1 and <alt>F1 are reserved for help access.

Reorderable

Allows the user to reorder the objects in the relation by drag and drop operations

Accept multiple selection

For nonexclusive relations, allow the user to select multiple items at once, by holding the *Ctrl* key while selecting using the mouse, or by using the *Shift* key and arrow keys.

Duplicable

When the user selects *Duplicate*, if the relation is exclusive, it will duplicate by cloning all the child objects of this member. For nonexclusive relations, it will only duplicate the objects links.

Note that for nonexclusive relation, if the *Unique* flag is set, the user will be able to link an object just once to the current relation. If not set, for *many to many* relation, the user will be able to add the same entity multiple times.

For nonexclusive relations, with a relatively small object base, you may set the **Use a combo box** and/or **Use editable box**, causing the relation to be rendered as a simple *RComboBox* (or *REditComboBox*).

16.4.16 Reference

References are virtual read-only relations, referring already existing relation. For example, Customer has Orders, linking to Product. If we want to see all the products ordered by a specific customers, on every order, a reference may be defined for customer, referring Orders/Products. This will generate a view, containing all the products from all the orders being made by the customer. It has the following specific properties:

Relation

The relation member withing the current entity to reference

Member

The member withing the relation to reference (may be of any type)

Path

Alternative version of Relation/Member properties. A path may be set in the similar syntax with the relation filters (see previous subsection). Unlink *Relation/Member* the Path property must reference only other relations.

Note that the *Relation/Member* properties cannot be used when *Path* is set.

16.4.17 Non homogeneous relation

A non homogeneous (or heterogeneous) relation can reference multiple entities simultaneously. This data type is to be avoided, due to limitations for reporting and data importing/exporting.

Reorderable
Allows the user to reorder the objects in the relation

Minimum items
Sets the minimum object count for the relation

Maximum items
Sets the maximum object count for the relation. If set to 0, it will be unlimited.

Mixed list level
Sets the minimum user level allowed to list the objects in the relation.

Enable send to option
Enable the "Send to" (for nonexclusive relations) or "New for" (for exclusive relations), in the related entity master view, enabling the user to add items th the current object without opening it first.

For each entity, in the mixed relation, the following properties may be set:

Relation with
Set the child entity. This property is mandatory.

Relation type
Set the type of the relation: *one-relation* for exclusive relation and *many-relation* for non-exclusive relations.

Minimum elements
Sets the minimum object count for the related entity

Maximum elements
Sets the maximum object count for the related entity. If set to 0, it will be unlimited.

16.5 Plug-ins

A special Gyro member type is the plug-in. A plug-in is created just like any other member, but it does not hold any actual data. Instead, it is rendered in an user data form. By default, Gyro has the following plug-ins:

EmailSender
Adds a send by e-mail button to a form

GoogleMaps
Shows a Google map, centered on a point described by a string member

GyroLineChart
Generates a line chart for a given relation member

WaterMark
Adds an water mark to an existing image member

WebBrowser
Navigates to an address specified by a string member

It is relatively easy to write your own plug-in. All plug-ins are located in the *plugins* directory, located in the GyroGers folder. Each plug-in must have a png icon (32x32 pixels), a parameter definition file, and a root directory, containing all the sources and resources needed.

For example, for a plug-in called MyPlugin, the following elements should be created: MyPlugin.png, MyPlugin.parameters, MyPlugin/include and MyPlugin/res folders.

The *MyPlugin.parameters* contains a line for each parameter in the following format:

```
Property name:mandatory|optional,
    string|boolean|number|entity|member| memberedit|option,value
```

As an example, consider the *EmailSender* plug-in, with the parameter definition:

```
1  Servername:mandatory,string,localhost
2  Port:mandatory,number,25
3  Username:optional,string
4  Password:optional,string
5  From:mandatory,memberedit
6  Email template:mandatory,longstring,"Hello world !!!"
7  Use members (separated by comma):optional,memberedit
8  To member:mandatory,memberedit
9  Subject member:mandatory,memberedit
10 Button caption:optional,string," by e-mail"
11 TLS:optional,boolean,false
12 TLS Trust file:optional,string,""
13 Authentification method:optional,option,
       off;on;plain;login;cram-md5;external;gssapi;scram-sha-1;digest-md5;ntlm
14 Send on write:optional,boolean,false
15 Add headers:optional,boolean,true
```

As a special case, the *optional* property type, has a list of all the possible values separated by “;” (see line 13). Also the *memberedit* type, allows the user to either select a member, or to type free text. The plug-in can then check if the member exists by using the *HasMember* function and see if it is a valid member name or some free text.

Each plug-in must defined a class, with the plug-in name, in the *include* directory. The class must implement the following methods:

static Query (string operation, target, prev_object, member1=”” ...)
This method is automatically called by Gyro at various operations, described by *operation*. Values for operation may be: “write”, called when an entity object is written into the database, “oninsert”, called when an object if first written into the database, “onupdate”, called when an object gets updated into the database, “delete”, called when deleted and “archive” called when the object is archived/unarchived.

static Create (_PlugInDataContainer Context, ContainerObject, OwnerForm, member1=””)
Called when the data form is created. The *Context* object describes the method events hooks for the plug-in (see bellow). The *ContainerObject* is a Concept container UI object holding the plug-in (it may be, for example an *RScrolledWindow*), and the *OwnerForm* is the form rendering the plug-in. This function must return an

plug-in instance (*new MyPlugin()*). *member1..N* parameters are string parameters, it the exact order of the plugin parameters file, containing the values selected by the programmer, as strings. Note that a boolean value will be set as the string “true” or “false”.

The *_PlugInDataContainer* class may have the following properties:

Set (Obj)
: Called when an entity object (*Obj*) is loaded into a data form. It allows the plug-in to configure itself for the loaded object.

Get (Obj)
: Called when an entity object is read from the form (for example when is saved). It gives the plug-in a chance to make modifications to that object.

boolean Validate (Obj, var error_string)
: It is called every time an object entity is validated. A plug-in may have its own validations, if needed. If the given entity object (*Obj*) is valid, this function must return true. If it is invalid, must return false and set the *error_string* parameter to a human-readable string describing the error.

UpdateFormula (Obj)
: Called when an entity object (*Obj*) needs recomputing of all the attached formulas.

For simplicity, the *WebBrowser* plug-in will be analyzed next. It has just two properties, defined in *WebBrowser.properties.*

```
URI member:mandatory,member
No URI callback:optional,boolean,true
```

And uses the following code:

WebBrowser/include/WebBrowser.con

```
include RWebView.con

class WebBrowser {
   private var WebView;
```

```
    private var URIMember;
    private var NoURICallback;

    public WebBrowser(Owner) {
        WebView=new RWebView(Owner);
        WebView.Show();
    }

    public Set(Obj) {
        // set !
        if (!URIMember)
            throw "WebBrowser plug-in error: no URI member set";

        var temp="";
        if (!GetMember(Obj, URIMember, temp))
            throw "WebBrowser plug-in error: no such member
                '${URIMember}'";

        temp=""+temp;

        if (!Pos(ToLower(temp), "http://"))
            temp="http://"+temp;

        WebView.Stop();
        WebView.URI=temp;

        return this;
    }

    public Get(Obj) {
        if (NoURICallback)
            return;

        if (!URIMember)
            throw "WebBrowser plug-in error: no URI member set";

        var uri=WebView.URI;
        if (uri) {
            if (!SetMember(Obj, URIMember, uri))
                throw "WebBrowser plug-in error: no such member
                    '${URIMember}'";
        }

        return this;
    }

    public Validate(Obj, var error_string) {
        error_string="";
        return true;
    }

    static Query(string operation, target, prev_object,
        URIMember="", NoURICallback="true") {
        // nothing
    }

    static Create(Context, ContainerObject, OwnerForm,
        URIMember="", NoURICallback="true") {
        var handler=new WebBrowser(ContainerObject);
        handler.URIMember=URIMember;
        if (NoURICallback=="true")
            handler.NoURICallback=true;
        else
            handler.NoURICallback=false;

        Context.Set=handler.Set;
        Context.Get=handler.Get;
        Context.Validate=handler.Validate;
        return handler;
    }
}
```

Note how the *URIMember* and *NoURICallback* parameters are given as strings.

Note that not all the features of the plug-ins are available on http:// and mobile applications, and may not be available on all platforms.

16.6 Conditional data

Each entity may have conditional data attached, defining what fields and/or categories are visible or not. The conditions are defined in XML, using a few simple tags.

The root object is called <*condition*> and it must have an *object* attribute, defining the object name. Each <*condition*> may have one ore more <*trigger*> nodes encapsulating a conditions. Each must be linked to an entity member, by setting the *property* attribute. The property will be set to the name of the member (natural or normalized), whose value must be monitored for changes.

Each <*trigger*> must contain one or more conditions. There are two types of conditions: <*if syntax=“...”* /> and <*case syntax=“..”*> .. </*case*>. The *syntax* attribute will contain a Concept condition that activates the trigger.

An activate trigger may either <*inactivate*> or <*hide*> a *field*(member) or a *category*.

Assuming an entity called Customer, having Type (Company, Person, Freelancer), Satisfaction level (Low, Medium, Hight) among usual members like VAT Number, Representative and so on, a conditional form will use:

```
<condition object="MyCustomer">
   <trigger property="Type">
      <if syntax="MyCustomer.Type==’Person’" />
      <!-- We could use multiple if tags -->
      <if syntax="MyCustomer.Type==’Freelancer’" />
      <hide category="Contacts" />
      <hide field="Representative" />
      <hide field="VAT number" />
      <!-- Inactivate the account field-->
      <inactivate field="Client account" />
   </trigger>
   <trigger property="Satisfaction level">
      <case syntax="MyCustomer.Satisfaction_level==’Low’">
         <hide field="Want newsletter" />
         <hide field="Want offers" />
         <hide category="Contacts" />
      </case>
      <case syntax="MyCustomer.Satisfaction_level==’High’">
         <hide field="Improvement suggestions" />
      </case>
   </trigger>
</condition>
```

This will cause the VAT number, Representative and all the members in the Contacts category to be hidden. The Client account member will still be visible, but rendered read-only for the user.

The *Satisfaction level* trigger will use mutual excluding conditions, for "Low" and "High", hiding fields accordingly.

16.7 Custom actions

Application custom actions are functions run by the end-user by pressing a button in the tool bar or menu bar. Each Action must reference a trigger function having the following prototype:

```
static boolean MyAction(Sender, ProgressForm, parameter = null);
```

The trigger function set to the function name, without parameters, for example *Misc::MyAction*. An action may have optional parameters, defined as a key-value array, or as a key-value array-returning function (including parameters), for example *Misc::GetSubActions(Connection)*. An example array would be *["Sub action 1" => 1, "Sub action 2" => 2]*. This will create an pop-up menu associated with the action, allowing the user to select one of the sub-action.

parameter is set only when the user selects a sub-action, and will have the value of the selected element from the action parameters array.

An action will have a status window with one or two progress bars (a primary and a secondary). The status history and progress are controlled via the *ProgressForm* parameter. If the function returns false, the progress form will be closed automatically. If it returns true, the user will have to close the window, allowing the read of all the status history (for example, errors).

The *CustomProgressForm* class has the following members:

Info (string message)
Adds an info message to the status history (prefixed by an info icon)

Warning (string message)
Adds a warning message to the status history (prefixed by a warning icon)

Error (string message)

Adds an error message to the status history (prefixed by an error icon)

Success (string message)
Adds an operation successfully message to the status history

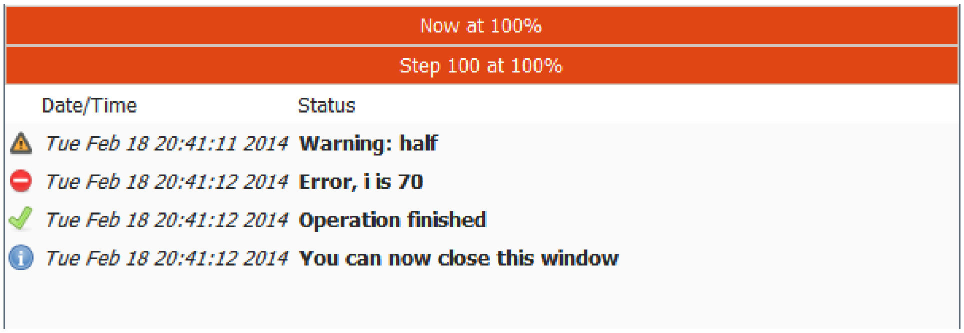
Clear (string message)
Clears all the status history

Progress (number fraction, string progress_caption)
Sets the primary progress bar value in the interval [0.00 .. 1.00], and sets the progress bar caption to given *progress_caption*.

Progress2 (number fraction, string progress_caption)
Sets the secondary progress bar value in the interval [0.00 .. 1.00], and sets the secondary progress bar caption to given *progress_caption*.

include/Misc.con

```
class Misc {
   static MyAction(Sender, ProgressForm, parameter = null) {
      // for localized messaging
      // var S = Sender.GetLang();
      // The database connection
      // var Connection = Sender.GetConnection();

      for (var i = 0; i < 100; i++) {
         ProgressForm.Progress((i + 1) / 100, "Now at ${i + 1}%");
         for (var j = 0; j < 100; j++) {
            ProgressForm.Progress2((j + 1) / 100, "Step ${i+1} at
                ${j + 1}%");
         }
         if (i==50)
            ProgressForm.Warning("Warning: half");
         if (i==70)
            ProgressForm.Error("Error, i is 70");
      }
      ProgressForm.Success("You can now close this window");
      ProgressForm.Info("You can now close this window");
      // the localized version
      // ProgressForm.Info(S << "You can now close this window");
      return true;
   }
}
```



Now at 100%

Step 100 at 100%

	Date/Time	Status
⚠	*Tue Feb 18 20:41:11 2014*	**Warning: half**
⛔	*Tue Feb 18 20:41:12 2014*	**Error, i is 70**
✓	*Tue Feb 18 20:41:12 2014*	**Operation finished**
ℹ	*Tue Feb 18 20:41:12 2014*	**You can now close this window**

Figure 16.3: Action status

This will result in a for shown to the user similar with the one in 16.3.

Custom action are useful when importing bulk data (for example from *.zip* files), processing or analyzing big datasets, or just a long-lasting operation.

For very long lasting operations, it is recommended to check for any messages pending in the application. The previous example should be rewritten as:

```
class Misc {
    static MyAction(Sender, ProgressForm, parameter = null) {
        for (var i = 0; i < 100; i++) {
            ProgressForm.Progress((i + 1) / 100, "Now at ${i + 1}%");
            for (var j = 0; j < 100; j++) {
                ProgressForm.Progress2((j + 1) / 100, "Step ${i+1} at ${j + 1}%");
            }
            if (i==50)
                ProgressForm.Warning("Warning: half");
            if (i==70)
                ProgressForm.Error("Error, i is 70");

            while (CApplication::MessagePending()) {
                if (CApplication::Iterate(Sender) == MSG_APPLICATION_QUIT) {
                    ProgressForm.Info("Operation interrupted by user ...");
                    return false;
                }
            }

        }
        ProgressForm.Success("You can now close this window");
        ProgressForm.Info("You can now close this window");
        return true;
    }
}
```

This will iterate the main queue, while processing the action function. Note that on disconnect, the *CApplication::Iterate* function will return MSG_APPLICATION_QUIT.

16.8 Advanced reports

GyroGears automatically generates some reports/exports based on the application specification. However, in some cases, special reports may be needed. For this, a GyroGears application may have advanced reports, explicitly defined by the developer. These reports are defined as XML, using a few tags. The XML root node is **<report>**. Each report has three sections:

<parameters>
Defines the report parameters (entered by the end-user)

<datasource>
Defines the data source (direct SQL query or Concept function)

<body>
Defines the layout of the report using tags similar to XHTML.

The report structure is:

```
<report>
    <parameter name="Parameter public name" type="string | number |
        boolean | date | datetime | combo | text | relation" with=""
        values="" default="" as="ParameterName"/>
    <datasource>
        <data query="select ... where name like ?" result="report1">
            <param>ParameterName</param>
        </data>
    </datasource>
    <body>
        .. report body ...
    </body>
</report>
```

All of the three sections are mandatory. The *parameter* tag will specify the parameters requested from the end-user. Each parameter must have at least the *name*, *type* and *as* attributes set. The *name* attribute is the parameter name as shown to the user, while the *as* attribute is the internal parameter name (as used by the developer). You should regard *name* as a caption. *type* specified the parameter type, and should be set to one of the values in the next table.

The Gyro report parameters may be:

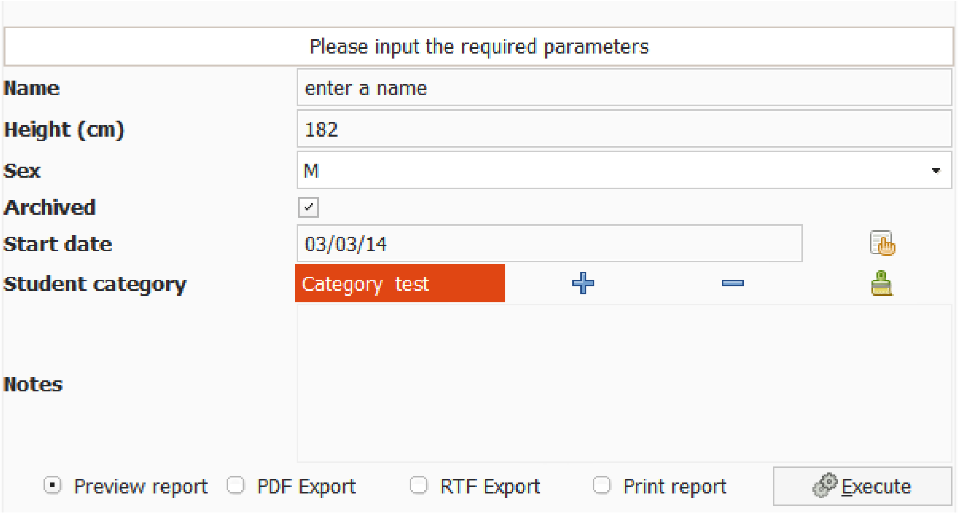
Type attribute	**Attributes**	**Notes**
string	default	The user will be asked to enter a string
text	default	The user will be asked to enter a long string
number	default	The user will be asked to enter a number
boolean	default	The user will be asked to check-/uncheck
date	default	The user will be asked to enter a date
datetime	default	The user will be asked to enter a date and a time
combo	values, default	The user will be asked to select from a combo box
relation	with	The user will be asked to select an entity object

The *default* attribute specifies a default value for the given parameter. For *combo* parameters, the values attribute keeps the list of values, separated by comma. For *relation* parameters with specifies the name of the target entity. All of the parameters will be returned as strings except for the *relation* parameters, returning an array containing the selected entity objects.

Assuming that an entity named *Category* is defined, the following parameters may be defined:

```
<report>
   <parameter name="Name" type="string" default="enter a name"
       as="Name" />
   <parameter name="Height (cm)" type="number" default="182"
       as="Height" />
   <parameter name="Sex" type="combo" default="M" values="M,F"
       as="Sex" />
   <parameter name="Archived" type="boolean" default="0"
       as="Archived" />
   <parameter name="Start date" type="date" default=""
       as="Start_date" />
   <parameter name="Student category" type="relation"
       with="Category" as="Category" />
   <parameter name="Notes" type="text" default="" as="Notes" />
   [..]
</report>
```

Resulting in the input form shown in figure 16.4.



Please input the required parameters

Name	enter a name
Height (cm)	182
Sex	M
Archived	☑
Start date	03/03/14
Student category	Category test
Notes	

● Preview report ○ PDF Export ○ RTF Export ○ Print report

Figure 16.4: Report parameters

The most important part of a report is the **<datasource>**. A data source may contain one ore more **<data>** queries, functions or query files.

The *data* node must specify one of the following attributes

query	A SQL query returning the data used in the report
query_file	Identical with *query*, but instead of directly specifying the query, a text file containing the query is used
function	A Concept function providing the data. The function must be given as delegate (without any parameters).

The data function prototype is:

```
static GetReportSource(Connection, array parameters);
```

Where *Connection* is the Gyro Application connection handle and parameters is an array of parameters specified by the **<param>** tag.

All **<data>** tags must have the *result* attribute set to an unique identifier. For example:

```
<report>
    <parameter name="Start date" type="date" default=""
        as="Start_date" />
    <parameter name="End date" type="date" default="" as="End_date"
        />

    <datasource>
        <data query="SELECT * FROM student WHERE Date >= ? AND Date
            <= ?" result="report1">
            <param>Start_date</param>
            <param>End_date</param>
        </data>
    </datasource>
    [..]
</report>
```

The same source may be defined using the a *data* function:

```
<report>
    <parameter name="Start date" type="date" default=""
        as="Start_date" />
    <parameter name="End date" type="date" default="" as="End_date"
        />

    <datasource>
        <data function="Misc::GetReportData" result="report1">
```

```
            <param>Start_date</param>
            <param>End_date</param>
        </data>
    </datasource>
    [..]
</report>
```

The corresponding data function:

```
class Misc {
    static GetReportData(Connection, parameters) {
        var data_start=parameters[0];
        var data_end=parameters[1];
        var[] result;
        // acquire data and return it as a matrix
        // containing one key-value array per row
        // for example:
        result = [
            ["column1" => "Value 1", "column2" => "Value 2"],
            ["column1" => "Value 1", "column2" => "Value 2"]
        ];
        // will return two rows, with two columns
        // (column1 and column2)
        return result;
    }
}
```

The last step in creating a report is the **<body>** definition. This defines the actual layout of the report.

The accepted *body* sub-tags are:

<p align="left|right|center|fill" margin="">

Creates a text paragraph. The *align* attribute specify the text align inside the paragraph. *justify* is an alias of *align*, having the same effect. The *margin* attribute specifies the text margin in points, percentage, cm or inches. For example:

```
<p align="10pt">This is a text paragraph</p>
```

<b>

Makes the text bold, for example:

```
<b>This text is bold</b>
```

<i>

Makes the text italic, for example:

```
<i>This text is italic</i>
```

<u>

Underlines the text, for example:

```
<u>This text is underlined</u>
```

<title align="left|right|center|fill" margin="">

Shows the text as the report tile, for example (in relatively bigger letters):

```
<title>This is the report tile</title>
```

<strong>

Makes the text bigger and bold, for example:

```
<strong>This is is important</strong>
```

<font family="" color="" bgcolor="" size="">

Shows text using the given font family, color, background color and size. Color must be in RGB hex format, prefixed by #.

```
<font family="Arial" color="#FF0000" size="22">This text is
    red</font>
```

<a href="">

Creates a hypertext link in the report.

```
<a href="devronium.com">Visit Devronium homepage</a>
```

<br />

Inserts a line break.

```
<p>This is a line<br/>And this is another line</p>
```

<img src="url" />

Insert an image into the document, specified by *url*.

```
<img src="res/logo.jpg"/>
```

<hr width="" height="" align="left|right|center"/>

Inserts a horizontal ruler. *width* and *height* may be in points or percentage. For example, a horizontal line aligned at right, with a width of 80% of its parent width, and a height of 2 points, cand be defined:

```
<hr width="80%" height="2pt" align="right"/>
```

<table width="" height="" border="" cellspacing="" cellpadding="">

Inserts a table. A table may have only *tr* and *thead* nodes as children. The border property must be in XSL-FO border format, for example border="2pt solid red" will create a red border line 2 points thick.

<tr bgcolor="">

Inserts a row in a table.

<thead bgcolor="">

Inserts a header row in a table (unlike *tr*, this row will be repeated on every page).

Each *tr* or *thread* may have only *td* as child. *td* defines a table cell. Note, that unlike the HTML *td* element, the report *td* cannot span over multiple columns and/or rows.

<td bgcolor="" border="" align="left|right|center" units="" width="">

The *td* tag will hold the actual data. Unlike HTML, the report data cells do not expand automatically to hold the text. You must specify either a fixed width or a number of *units* used. For example, if a *td* uses one unit (*units*="1") and another one uses two units, it would make the second cell twice as big as the first unit. *td* also inherits all of the *p* attributes.

A simple table could be created as:

```
<report>
[...]
<body>
  <table width="100%" border="2pt solid red">
    <thead bgcolor="#A0A0A0">
      <td align="left" units="1">Index</td>
      <td align="center" units="3">Name</td>
      <td align="right" units="2">Value</td>
    </thead>
    <tr>
      <td align="left">1</td>
      <td align="center">Eddie</td>
      <td align="right">1000000</td>
    </tr>
    <tr>
      <td align="left">2</td>
      <td align="center">Maria</td>
      <td align="right">2000000</td>
    </tr>
  </table>
</body>
</report>
```

The result is shown in figure 16.5.

<pie width=“” height=“” from=“” field=“” value=“” />

Inserts a data-aware pie chart using the given data as source (*from*), mapping the values of *field* to *value* attribute. In other words, the *field* attribute sets the legend (caption) and the *value* attribute selects the actual column value. Assuming that an entity called Student is defined, and it has at least one member, called Sex (combo or radio, selecting M or F), a minimal example, without any parameters would be:

```
<?xml version="1.0"?>
<report>
    <datasource>
        <data query="SELECT COUNT(*) AS cnt, sex FROM student
            GROUP BY sex" result="report1" />
    </datasource>
    <body>
        <title>Male/Female ratio of students</title>
        <pie width="500" height="300" from="report1"
            field="sex" value="cnt" />
    </body>
</report>
```

The output is shown in figure 16.6.

<chart type="area|line" width="" height="" from="" field="" />

Inserts a data-aware line chart using the given data as source (*from*). The *from* attribute is the legend or series names. Each chart must have at least one *carrier* child node.
<carrier from="" field="" axis="" color=""> describes the line(s) available on the chart. The *from* attribute refers the data source, the *field* attribute gives the y coordinate and the *axis* attribute gives the x coordinate.

```
<?xml version="1.0"?>
<report>
    <datasource>
        <data query="SELECT TrafficDirection, Day, Duration
            FROM phone_call" result="report1" />
    </datasource>
    <body>
      <chart type="area" width="1000" height="500"
          from="report1" field="TrafficDirection">
        <carrier from="report1" field="Duration" axis="Day"/>
      </chart>
    </body>
</report>
```

The output is show in figure 16.7.

<if condition="">

Checks for a specific Concept condition, for example:

```
<report>
    [..]
    <if condition="Misc::CheckCondition()">
        <notify>Warning: Condition is not met</notify>
    </if>
    [..]
</report>
```

Where CheckCondition is a function implemented by the developer. If Misc::CheckConditions returns non-zero, the given message will be shown to the user.

<abort>
Aborts the report generation with the given message, for example:

```
<report>
    [..]
    <if condition="Misc::CheckCondition()">
        <abort>The report could not be generated</abort>
    </if>
    [..]
</report>
```

<notify>
Shows a message to the user, without aborting the report generation.

<xsl-fo> xmlns:fo=”http://www.w3.org/1999/XSL/Format” xmlns:xsl=”http://www.w3.org/1999/XSL/Transform”
The *xsl-fo* tag enables the use of actual XSL-FO code. The data set is automatically given by Gyro as the input XML.

```
<report>
    <datasource>
        <data query="SELECT * FROM student" result="report1" />
    </datasource>
    <body>
        <xsl-fo xmlns:fo="http://www.w3.org/1999/XSL/Format"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
          <fo:table>
            <fo:table-column column-width="25mm"/>
            <fo:table-column column-width="50mm"/>
            <fo:table-column column-width="25mm"/>

            <fo:table-header>
              <fo:table-row>
                <fo:table-cell>
                  <fo:block font-weight="bold">Number</fo:block>
                </fo:table-cell>
                <fo:table-cell>
                  <fo:block font-weight="bold">Student
                      name</fo:block>
                </fo:table-cell>
                <fo:table-cell>
                  <fo:block font-weight="bold">Sex</fo:block>
                </fo:table-cell>
              </fo:table-row>
            </fo:table-header>

            <fo:table-body>
              <xsl:for-each
                  select="ReportXML/Data/report1/array">
                  <fo:table-row>
                    <fo:table-cell>
                      <fo:block><xsl:value-of
                          select="position()"/></fo:block>
                    </fo:table-cell>
                    <fo:table-cell>
                      <fo:block><xsl:value-of
                          select="name"/></fo:block>
                    </fo:table-cell>
                    <fo:table-cell>
                      <fo:block><xsl:value-of
                          select="sex"/></fo:block>
                    </fo:table-cell>
                  </fo:table-row>
              </xsl:for-each>
            </fo:table-body>
          </fo:table>
        </xsl-fo>
    </body>
</report>
```

The report input XML generated by Gyro will always have the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<ReportXML>
  <Data>
    <report1>
      <array>
        <id>2</id>
        <name>Maria</name>
        <sex>F</sex>
      </array>
      <array>
        <id>3</id>
        <name>John</name>
        <sex>M</sex>
      </array>
    </report1>
  </Data>
</ReportXML>
```

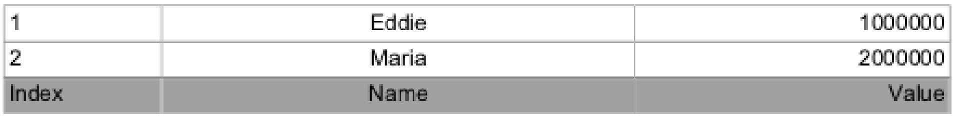
XSL-FO code may call any function implemented in the *Misc* class, for example, line 33 of the given report may be replaced with:

```
<fo:block><xsl:value-of
    select="csp:Misc.ToUpper(name)"/></fo:block>
```

Where ToUpper is defined in Misc as:

```
class Misc {
    static ToUpper(string name) {
        return ToUpper(name);
    }
}
```

The ”csp:” prefix, short from *Concept Server Page*, will cause a call to the given Concept function. All of the function parameters will be given as strings. The function may return a string, number or an array.



1	Eddie	1000000
2	Maria	2000000
Index	Name	Value

Figure 16.5: Report table example

16.9 Sub-applications

A Gyro application may have one ore more sub-applications or child applications. Each child application is an independent application, using the same database as the mother application. The developer can which entities and/or members, actions, events and reports are available to the child application. All the child applications are generated in the solution

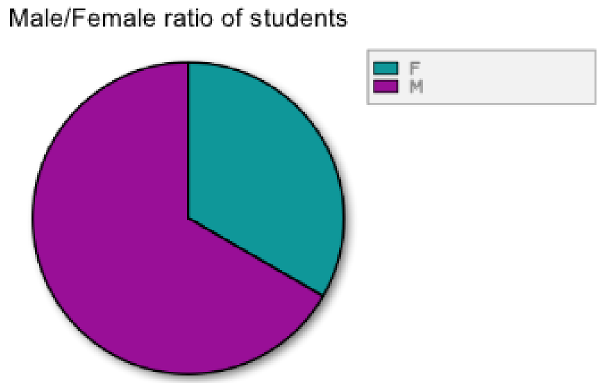


Figure 16.6: Report pie chart

sub-folder of the mother application. Gyro will also generate an interface, allowing the user to select a child application or the full application.

Each child application has some specific properties, with identical behavior as the mother application.

The only child application specific property is **Single entity mode**. When this is set to refer an existing entity, the child application will automatically open the master view of the selected entity, in full screen, without allowing the user to open any other views.

16.10 Applications diagrams

Gyro can generate a diagram describing the application. A diagram will contain basic information about the entities, members and relation. Diagrams are useful when analyzing and discussing data models with other developers and/or clients. A sample diagram is shown in figure 16.9. There are a few conventions used on these diagrams:

light red background
The member using the light red is the representative(or quick search) member of the entity.

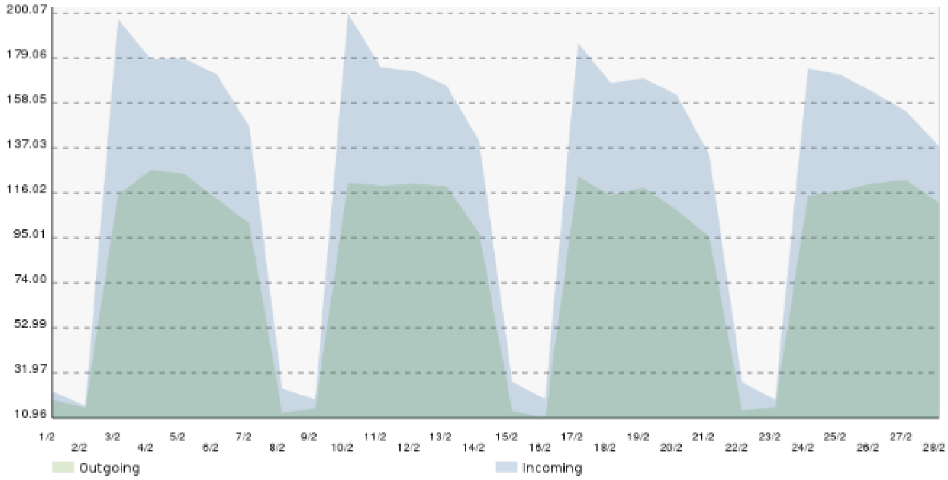
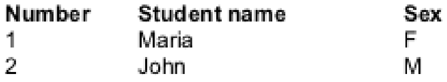


Figure 16.7: Report chart



Number	Student name	Sex
1	Maria	F
2	John	M

Figure 16.8: XSL-FO example output

light green background
The member is a relation.

dark red dot
The member is a mandatory.

line ending in ▷, connecting a member with an entity
The member describes an exclusive relation with an entity.

line ending in ◁, connecting a member with an entity
The member describes a non-exclusive relation with an entity.

16.11 Touch screens

By enabling the touch screen support for a gyro application, by setting the "Optimize for touch screen" application property to *Yes*, either a numeric pad or an on-screen keyboard will be provided to the end-user. Also, setting this property, will cause Gyro to generate slightly larger buttons for easy touching.

The on-screen keyboard (figure 16.10) will be available by pressing a special icon contained in the edit field, giving the user a chance to use the native operating system keyboard. The numeric keypad (figure 16.11) however, will be displayer just by focusing the data field.

16.12 Phone and tablet interface

If "Generate mobile version" property is set to *Yes*, Gyro will generate a mobile version of the application that can be opened by the Concept Client for Android and iOS. At the time this book was written, there are 3 mobile client version, running on Android 2.3.x, Android 4.x or iOS 7.0 or grater. There are some plans for supporting Windows Phone and some Smart TV operating systems.

The Gyro application mobile version has a different access point from the standard desktop version. Assuming that an application names CRM has the following entry point:

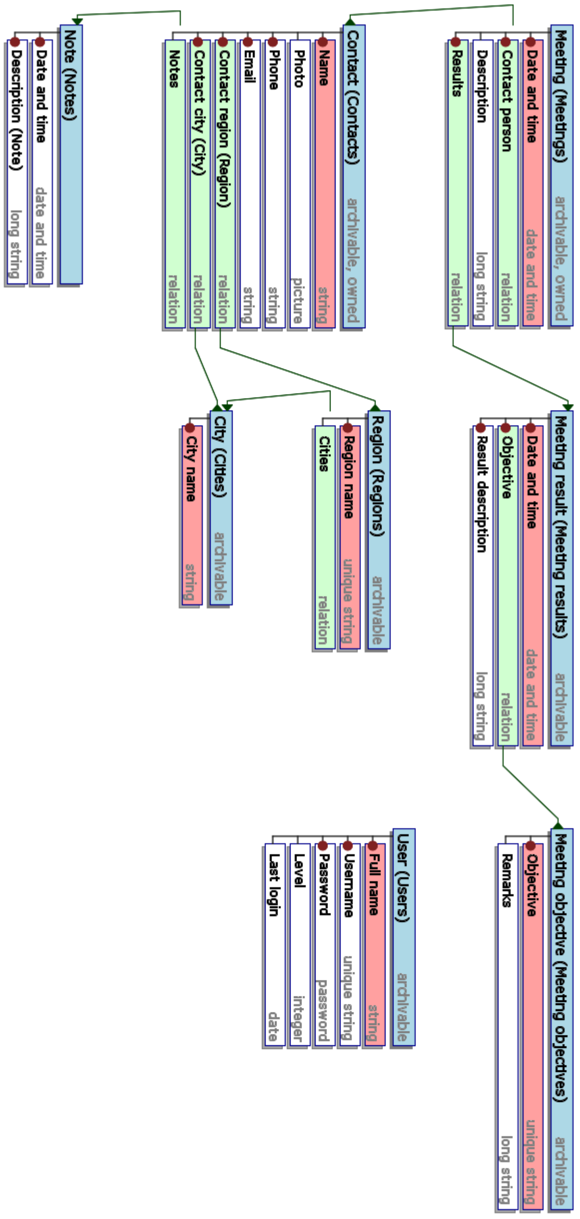

Meeting (Meetings) archivable, owned
Date and time date and time
Contact person relation
Description long string
Results relation
Contact (Contacts) archivable, owned
Name string
Photo picture
Phone string
Email string
Contact region (Region) relation
Contact city (City) relation
Notes relation
Note (Notes)
Date and time date and time
Description (Note) long string
Meeting result (Meeting results) archivable
Date and time date and time
Objective relation
Result description long string
Region (Regions) archivable
Region name unique string
Cities relation
City (Cities) archivable
City name string
Meeting objective (Meeting objectives) archivable
Objective unique string
Remarks long string
User (Users) archivable
Full name string
Username unique string
Password password
Level integer
Last login date

Figure 16.9: Gyro diagram

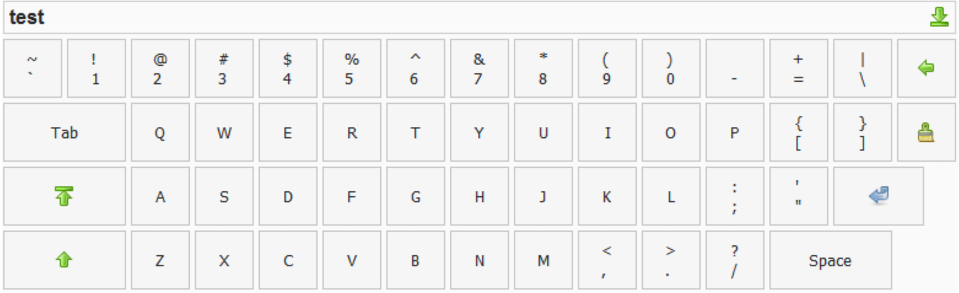

test

Figure 16.10: On screen keyboard

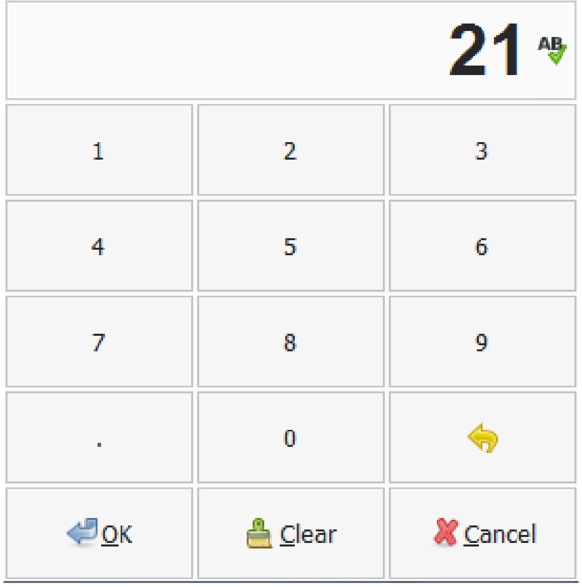

21

Figure 16.11: Numeric keypad

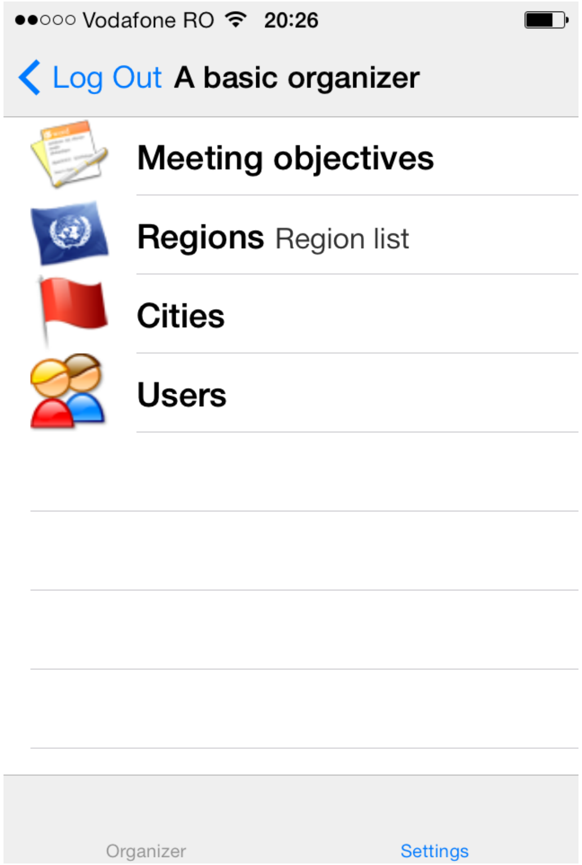


Figure 16.12: Gyro application running on iOS 7

```
concept://serverhost/MyProjects/CRM/CRM.con
```

The entry point for the mobile application is:

```
concept://serverhost/MyProjects/CRM/m.con
```

Note that m.con, located in the application directory must be compiled manually, by running (in the generated application directory):

```
accel m.con
```

The application can then be loaded on an iPhone/iPad or Android phone or table by specifying the link (replacing *serverhost* with your server host name or ip). The application described by the diagram shown in figure 16.9 should look similar to figure 16.12.

16.13 Application .ini files

Each Gyro-generated application will have an *.ini* file in the root directory, for setting up the database connection.

The ini file is named after the selected database engine, for example, for MySQL it is called MyDataBase.ini, for SQLite it is called SLDataBase.ini, for PostgreSQL PQDataBase.ini, Firebird is FBDataBase.ini, Mongo is MongoDataBase.ini and ODBC it is simply DataBase.ini.

An example ini generated by Gyro for a MySQL-based application, would be:

MyDataBase.ini

```
[DataSource]
DataBase      =   "BasicOrganizer"
Host          =   "localhost"
Port          =   3306
Schema        =   ""
Username      =   "root"
Password      =   ""

; Folder used to keep large files, if necessary
LargeFilesFolder= "datastore"
IndexFolder   =   "dataindex"
CacheFolder   =   "cache"

; Memcached server
UseSuperCache =   1
MemCachedServer = "localhost"
MemCachedExpire = "3600"

; Spell checker
Language      =   "en_US"
```

Where *DataBase*, *Host*, *Port*, *Username* and *Password* describe the database server connection.

LargeFileFolder sets the directory to be used for storing blobs, if the developer opted for storing files and multimedia content on disk instead of the database.

IndexFolder sets the directory for the Xapian database, if enabled. Cache folder is the directory to be used for image caching (only for web applications).

If *UseSuperCache* is set to 1 (0 is for disabled), the application will cache its data on a *memcached* server located at the address specified by *MemCachedServer*. The *MemCachedExpire* sets the data timeout.

Finaly, if spell checker is used, the *Language* parameter sets the used dictionary. Note that Gyro provides only the en_US, en_GB and ro_RO dictionaries. For other languages you must find an appropriate dictionary and put the files into the *res* directory of your application (an .aff file and a .dic file, for example de_DE.aff and de_DE.dic).

16.14 Data migration

In some case, a database server will need replace. For example, an application using MySQL may need migrating to a PostgreSQL server or Firebird. The application can be simply regenerated, by selecting a new *database connection method* and a new set of *database rules*. This will change the database server used by the application, but will not migrate the data contained on the old server.

For that, a directory called *migrate* is created on the application root, containing a few scripts for migrating data from the current database to a new supported database engine. Note that migration from SQL to NoSQL or vice versa is not yet supported.

The directory will contain multiple sets of scripts and ini files for various database engines. For example, assuming that an application is using MySQL and needs to be ported on PostgreSQL, *migrate/MigratePQDataBase.ini* and *migrate/MigratePostgreSQL.con* will be used. The ini file will point to the new database in which the current data, referred by the application ini file, located in the application root - in our case MyDataBase.ini, is to be written. The developer should set the new server, database name, user and password for the new database, and then simply run *MigratePostgreSQL.con* and wait for the data to be copied.

Note that for PostgreSQL and Firebird the ID sequences/generators for each entity must be initialized manually. Also, the table structures in the new database must exist and contain no elements. For this, in many cases, the developer will need to generate and run the new application before migrating the data, resulting in data structure creation. In this case, Gyro will keep the previous data migration scripts in a directory called *migrate.previous*. This being the case, the scripts in *migrate.previous* should be used instead of the ones in *migrate*.

16.15 Web interface

If the "Generate traditional Web 2.0 interface" is set to *Yes*, GyroGears will generate a web application that can be opened from a web browser. Note that if the "Generate Web 2.0 sign up" is also set to *Yes*, a sign-up script will be generated, allowing anonymous users to register themselves. The web application root will be in the *web20* directory of the generated Concept application. Unlike the concept:// application, multiple files will be created. For compiling a web application, the script *Application.Make.con* must be run, for example:

```
concept Application.Make.con
```

The ini file used by the web application is located in the *web20/ini* subdirectory, instead of the root directory. As a note, is better to limit via the Apache *.htaccess* or similar mechanisms the access to the *ini* directory for the web server.

This will compile all the sources generated by Gyro for the web application. All the application web scripts are based on XML, an XSLT template and an XSLT processor for producing XHTML output.

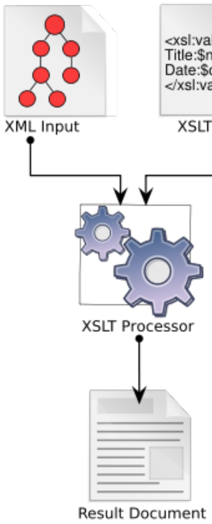


Figure 16.13: XML and XSLT producing XHTML (Attribution: Dreftymac at en.wikipedia)

A Gyro web application may be accessed by opening from a web browser:

```
http://hostname/ApplicationName/web20/
```

Assuming that the web server document root is set to Concept's *MyProjects* directory. *hostname* should be replace with the web server's host name, for example, localhost:8080, and ApplicationName is the Gyro application name, for example, *CRM*. In this case, the link should be:

```
http://localhost:8080/CRM/web20/
```

For each entity, gyro will generate two *.csp* (Concept Server Page) scripts, called **EntityName**.csp and **EntityName**List.csp, where *EntityName* is the normalized name of the entity. For example, for an entity called Contact, Contact.csp and ContactList.csp will be generated in the *web20* directory. Also, in the *web20/tpls*, three XSLT templates will be generated, EntitName.edit.xslt (the editing template), EntityName.xslt (object view template), EntityName.list.xslt (the object view template). These are automatically generated by Gyro, using a minimal template, but in practice, the XSLT will be created by a front-end web developer and/or an web designer.

*List.csp is the master object view equivalent from the concept:// application. Every *List.csp script has the following parameters (that can be sent via GET or POST):

q

The search query used for retrieving the objects. This will perform a database search or an probabilistic(Xapian) search on the entity objects, using the query string.

u

Filter the data using the given user id. If *u* is set to -1 (default), objects from all the user will be returned.

noheader = 0|1

If set to 1, will skip the Javascript headers for the generated document.

page

Selects the given page, starting from 1.

pagesize

Sets the number of objects per page (default is Page size property of the entity).

json = 0|1

If set to 1, it will return the view in JSON format.

format

This parameter is transparently passed to the XSLT template. This will enable the developer to generate different documents based on the given format. However, *format=xml* is reserved. In this case, the script will return the input XML, without doing any XSLT processing. This is very useful for debugging or creating APIs for third-party software.

sort

Sets the field name used for sorting, if the “Use in sorting” flag is set. For example, if you want to sort objects by a field called Email_address, *sort=Emal_address* should be used

desc = 0|1

If set to 1, the objects will be sorted in descending order.

xslt

Sets an alternate custom template to be used, for example, *xslt=tpls/CustomTemplate.xslt* will perform the transformations using the given file.

__REDIRECT

Force a redirect after a given operation succeeds.

parent_level

If set to 1 ore more, the retrieved objects will contain references to the parent object (if any). If set to 2, it will also retrieve the parent's parent, and so on.

related_level

If set to 1 ore more, the retrieved objects will contain references to all its child objects (if any). If set to 2, it will also retrieve the children of each child, and so on.

related_pagesize

Sets the child relation page size (useful if related_level is set to 1 ore more). Default is the related entity page size (usualy 50 objects).

related_page

Selects the child page, starting from 1.

fullusers = 0|1

Instead of having references for each owned object to the user id, it will return a full record referencing the owner user, containing all the user information, except password.

just

Optinal parameter for selecting the fields to be retrieved for the given list. If is not set, all the fields fill be retrieved. All the objects must reference the entity normalized name and member name, separated by dot, in a list separated by comma. For example, for an entity called Contact (Name, Email, Address, Notes), if only the *Name* and *Email* fields are wanted, *just* can be set to *just=contact.name,contact.email.* This is especially useful when *related_level* is set to 1 ore more, and not all the relations are needed, reducing the document generation time.

filter1,op1,val1...filterN,opN,valN

filters enable the use of advanced filters (for members having the *advanced search* flag set to true). Each filter must be given as a filter, op, val pair. For example, assuming that an entity having a member called *Date* is defined, and in a page the objects with date between "2014-01-01" and "2014-02-01" is needed. For this, the following parameters could be used:
filter1=date&op1=%3E%3D&value1=2014-01-01
&filter2=date&op2=%3C&value1=2014-02-01

Note that %3E%3D is the URL encoded version of >= and %3C is the encoded version of <.

For relational filters, value1 may contain multiple values separated by comma, for example: *filter1=city&op1=%3E&value1=1,2,3* it will search of all the objects having the *City* member related with City objects having 1, 2 or 3 as ID.

Any other parameter will be transparently passed to the XSLT template (a developer may add any number of parameters).

Assuming that we defined an entity called Contact, a request to:

```
http://localhost:8080/Organizer/web20/ContactList.csp?format=xml
```

Will output:

```
<?xml version="1.0" encoding="UTF-8"?>
<HTMLContainer>
  <LoggedIn>0</LoggedIn>
  <UID>-1</UID>
  <ScriptID>ContactList</ScriptID>
  <Username/>
  <PrevLink/>
  <NextLink/>
  <PageLink>
    <number>0</number>
  </PageLink>
  <Page>1</Page>
  <TotalPages>1</TotalPages>
  <PageSize>50</PageSize>
  <Criteria/>
  <Sort/>
```

```
<Desc>0</Desc>
<Count>2</Count>
<Format>xml</Format>
<RelatedPage>1</RelatedPage>
<RelatedPageSize>50</RelatedPageSize>
<Response/>
<Redirect/>
<Template>tpls/Contact.list.xslt</Template>
<Caller/>
<IsChildOf/>
<NoHeader>0</NoHeader>
<Message/>
<Op/>
<LinkHint/>
<Key/>
<Title/>
<Settings/>
<Data>
  <Contact>
    <__DBID>4</__DBID>
    <__OUID>1</__OUID>
    <__MUID>1</__MUID>
    <__VIEWINDEX>1</__VIEWINDEX>
    <__EVENINDEX>1</__EVENINDEX>
    <Name>Mister Spock</Name>
    <Photo/>
    <Photo_filename>clip1394010172.jpeg</Photo_filename>
    <Photo_thumbnail/>
    <Photo_preview/>
    <Phone>+888135116829</Phone>
    <Email>spock@vulcan.vl</Email>
    <Contact_region>Region 2</Contact_region>
    <Contact_city>My city</Contact_city>
  </Contact>
  <Contact>
    <__DBID>3</__DBID>
    <__OUID>1</__OUID>
    <__MUID>1</__MUID>
    <__VIEWINDEX>2</__VIEWINDEX>
    <__EVENINDEX>0</__EVENINDEX>
    <Name>Zoe Doe</Name>
    <Photo/>
    <Photo_filename>Picture 321.jpg</Photo_filename>
    <Photo_thumbnail/>
    <Photo_preview/>
```

```
      <Phone>+22 12312312</Phone>
      <Email>jane@mail.com</Email>
      <Contact_region>Region 2</Contact_region>
      <Contact_city>My city</Contact_city>
    </Contact>
  </Data>
  <SecondaryData/>
  <QueryParameters>
    <format>xml</format>
  </QueryParameters>
</HTMLContainer>
```

Note that the <Settings> will contain references to all the entities with the “Is settings entity” set to *Yes*.

The resulting web page can be requested by using:

```
http://localhost:8080/Organizer/web20/ContactList.csp
```

The output is shown in figure 16.14. Note that the application uses a standard template. For production, an application custom template is recommended. More information about XSLT are widely available on the Internet, for example http://www.w3schools.com/xsl/.

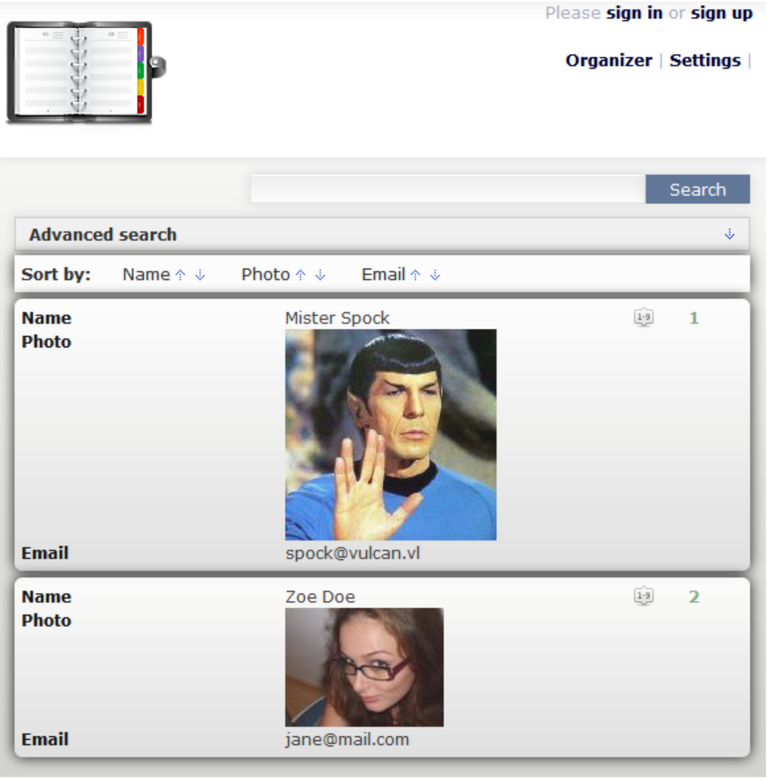
The generate XSLT template, tpl/Contact.list.xslt can be used as a template for changing the design of the web page. It can be either be replaced with the new template or use the xslt paramter:

```
http://localhost:8080/Organizer/web20/ContactList.csp?xslt=tpls/template.xslt
```

A call to a concept function defined in *Utils*, *WebUtils*, *Misc*, *Main* or specific *WU_** classes can be made from an XSLT template. WU is short from Web Utils.

For example (for a complete list, check the *Utils.con* file, located in the generate application *web20/include* directory):

```
// forces a log out operation
csp:WebUtils.LogOut()
// returns the current date as a string
csp:Utils.DateNow()
```



Please **sign in** or **sign up**

Organizer | **Settings** |

Advanced search

Sort by: Name ↑ ↓ Photo ↑ ↓ Email ↑ ↓

Name	Mister Spock	1
Photo		
Email	spock@vulcan.vl	

Name	Zoe Doe	2
Photo		
Email	jane@mail.com	

Figure 16.14: ContactList.csp output in an web browser

For WU_* specific function, the following functions are useful in templates:

```
csp:WU_EntityName.View(page=0, pagesize=50, criteria="",
    sort_field="", desc=0, UID=-1, GID=-1, show_archive=0,
    no_blobs=2, do_format=1, do_html_escape=1);
```

This will return an array containing all the elements matching the given criteria.

For relational members:

```
csp:WU_EntityName.GetMemberName(page=0, pagesize=50, criteria="",
    sort_field="", desc=0, UID=-1, GID=-1, show_archive=0,
    no_blobs=2, do_format=1, do_html_escape=1);
```

Entity should be replaced with the entity normalized name, and MemberName with the member normalized name.

For example, if entity is called *Contact* and a relation member to some notes is called Notes, the function names will be *csp:WU_Contact.View()* and *csp:WU_Contact:GetNotes(id)*.

Note that when invoking a Concept function from XSLT, you cannot use **true** or **false**. Instead you should use 1 for true and 0 for false.

The *EntityName.csp* script will view or edit a specific object. The GET/POST parameters are:

id
: The id of the object to show.

__REDIRECT
: Force a redirect after a given operation succeeds.

op = add|new|del|arc|unarc
: Performs a specific operation on the given object identified by *id*, assuming that the logged user has enough rights.

  op values are:

  add Updates the current object into the database. The member values are sent via GET or POST to the *EntityName.csp* script, and must use the normalized member names (case-sensitive). For example, if an entity has two fields, *Name* and *E-mail*, the application must send via GET or POST variables named "Name" and "E_mail" having the new value that needs to be updated.

new Creates a new object without writing it in the database

del Deletes the object specified by *id*

arc Archives the object specified by *id*

unarc Unarchives the object specified by *id*

After a successful operation, the user will be redirected to the address specified by the __REDIRECT parameter, if set.

k

For objects having a representative member marked as unique, *k* (key) will retrieve an object based on the member value, instead of using the *id*. For example, if an entity called Contact will have a representative member called Name that is also marked as unique, instead fo loading Contact.csp?id=4 a call to Contact.csp?k=Mister%20Spock will result in opening the same record, without exposing the ip in the address.

w

A reference to the parent script, for example *Parent.csp*. This is useful only when a link to the parent is needed in a form.

p

The parent *id*, if needed in the XSLT.

with_links

Add references to current logged in user in the input XML.

noheader = 0|1

If set to 1, will skip the Javascript headers for the generated document.

format

This parameter is transparently passed to the XSLT template. This will enable the developer to generate different documents based on the given format. However, *format=xml* is reserved. In this case, the script will return the input XML, without doing any XSLT processing. This is very useful for debugging or creating APIs for third-party software.

xslt
: Sets an alternate custom template to be used, for example, *xslt=tpls/CustomTemplate.xslt* will perform the transformations using the given file.

related_level
: If set to 1 ore more, the retrieved objects will contain references to all its child objects (if any). If set to 2, it will also retrieve the children of each child, and so on.

related_pagesize
: Sets the child relation page size (useful if related_level is set to 1 ore more). Default is the related entity page size (usualy 50 objects).

related_page
: Selects the child page, starting from 1.

fullusers = 0|1
: Instead of having references for each owned object to the user id, it will return a full record referencing the owner user, containing all the user information, except password.

just
: Optinal parameter for selecting the fields to be retrieved for the given list. If is not set, all the fields fill be retrieved. All the objects must reference the entity normalized name and member name, separated by dot, in a list separated by comma. For example, for an entity called Contact (Name, Email, Address, Notes), if only the *Name* and *Email* fields are wanted, *just* can be set to *just=contact.name,contact.email.* This is especially useful when *related_level* is set to 1 ore more, and not all the relations are needed, reducing the document generation time.

A call to:

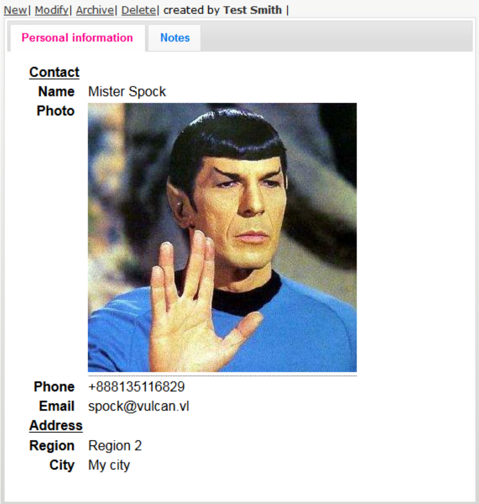
```
http://localhost:8080/Organizer/web20/Contact.csp?id=4&format=xml
```

Will result in:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<HTMLContainer>
  <LoggedIn>0</LoggedIn>
  <UID>-1</UID>
  <ScriptID>Contact</ScriptID>
  <Format>xml</Format>
  <RelatedPage>1</RelatedPage>
  <RelatedPageSize>50</RelatedPageSize>
  <Template>tpls/Contact.xslt</Template>
  <NoHeader>0</NoHeader>
  <Op/>
  <Title>Mister Spock</Title>
  <CanAdd>0</CanAdd>
  <CanModify>0</CanModify>
  <CanDelete>0</CanDelete>
  <CanArchive>0</CanArchive>
  <Data>
    <__DBID>4</__DBID>
    <__OUID>1</__OUID>
    <__MUID>1</__MUID>
    <__VIEWINDEX>1</__VIEWINDEX>
    <Name>Mister Spock</Name>
    <Photo/>
    <Photo_filename>clip1394010172.jpeg</Photo_filename>
    <Photo_thumbnail/>
    <Photo_preview/>
    <Phone>+888135116829</Phone>
    <Email>spock@vulcan.vl</Email>
    <Contact_region>Region 2</Contact_region>
    <Contact_city>My city</Contact_city>
  </Data>
  <SecondaryData/>
  <QueryParameters>
    <id>4</id>
    <format>xml</format>
  </QueryParameters>
</HTMLContainer>
```

And the corresponding web page:

```
http://localhost:8080/Organizer/web20/Contact.csp?id=4
```



New| Modify| Archive| Delete| created by Test Smith |

Personal information | Notes

Contact

Name	Mister Spock
Photo	
Phone	+888135116829
Email	spock@vulcan.vl

Address

Region	Region 2
City	My city

Figure 16.15: Web object instance output

Resulting in the output shown in figure 16.15.

If the editing template was to be selected (or the modify button pressed), assuming that the user is logged in and has editing rights, a call to:

```
http://localhost:8080/Organizer/web20/Contact.csp?id=4&xslt=tpls/Contact.edit.xslt
```

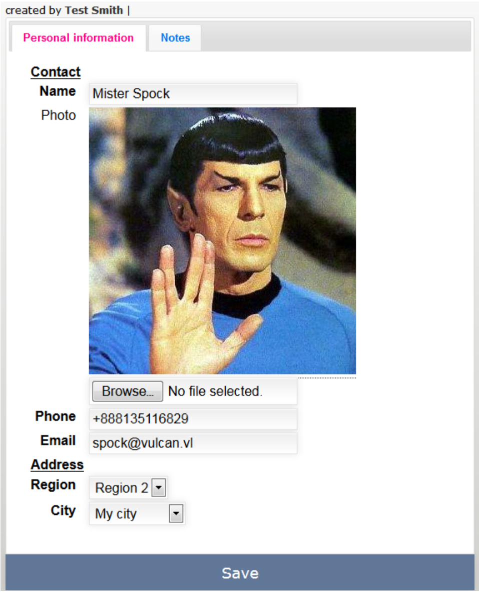
Will result in the output shown in figure 16.16.

For file, multimedia and picture members, Gyro will generate an additional script, called get*EntityMember*.csp, where *Entity* is the entity's normalized name and *Member* the member normalized name.

The GET/POST parameters are:

id
: The *id* of the object. Applies to file, multimedia and picture.

thumb = 0|1
: If set to 1, it will return the thumbnail picture. Applies to multimedia and picture.



created by Test Smith |

Contact

Name: Mister Spock

Photo

No file selected.

Phone: +888135116829

Email: spock@vulcan.vl

Address

Region: Region 2

City: My city

Figure 16.16: Object editing

preview = 0|1
: If set to 1, it will return the preview picture. Applies to multimedia and picture.

w

Returns the picture at the given width. Applies to multimedia and picture.

h

Returns the picture at the given height. Applies to multimedia and picture.

crop = 0|1

If set to 1, the picture will be cropped in order to have the requested width and height (w and h). If set to 0 (default), the picture will be resized without deforming it (best-fit). Applies to multimedia and picture.

Note that for w, h and *crop* parameters to work, special directories must be created in the *cache* directory, giving permissions to the system to process the image at the given size. This is for avoiding attacks that could request a huge amount of pictures at various size, overloading the CPU.

The default cache directory is *cache*, in *web20*. This directory could be changed by editing the application *.ini* file and changing the *CacheFolder* to a new relative location.

Assuming that the entity *Contact* has a picture member called *Photo*. The script name will be getContactPhoto.csp. If images of w=320 and h=240 or w=640 and h=480 pixels in size are needed, in the *cache* directory, the following directories should exist:

```
cache/Contact/Photo/320x240/
cache/Contact/Photo/640x480/
```

Also, the web server must have rights to write in those directories. Note that for default sizes (like thumb and preview, the cache directory is not used).

For multimedia members, a video player script is generated by Gyro, called Show*EntityMember*.csp, where *Entity* is the entity's normalized name and *Member* the member normalized name. For now, it uses a flash player, but it will be replaced by a HTML5 video player based on the <video>.

The GET/POST parameters are:

id
The *id* of the entity having the multimedia member.

noheader = 0|1
If set to 1, will skip the Javascript headers for the generated document.

xslt
Sets an alternate custom template to be used, for example, *xslt=tpls/CustomTemplate.xslt* will perform the transformations using the given file.

format
This output format (similar to the format parameter of the other scripts).

Index