General Manual

Table of Contents

About This Document.... 4
Why You Should Read It.... 4
Note Of Thanks.... 5
About The Author.... 5
Supporter.... 5
Tell A Friend!.... 5
Objective-Basic Needs Your Help!.... 5
Give Us Feedback.... 6
New Set Of Manuals.... 6
Installation.... 6
Introduction.... 7
Cocoa.... 7
What is a Computer Program?.... 8
Programming Languages.... 9
History of the Name.... 9
All Kinds of BASIC.... 10
Compilers and Interpreters.... 10
The Programming Process.... 11
Attack of the Bugs.... 11
How much do I need to know to use it?.... 12
Introduction to the Objective-Basic programming language.... 12
- What is Objective-Basic?.... 12
- Briefly and well in a sentence.... 13
- Objective-Basic's Past, Present and Future.... 13
- What is still needed to know about Objective-Basic?.... 13
- Objective-Basic is object-oriented.... 13
- Stable.... 14
- Objective-Basic is Fast.... 14
- Rapid application development.... 14
- How Can I Get Objective-Basic?.... 15
- On which platform is Objective-Basic available?.... 15

Software development with Objective-Basic.... 15
- Event controlled programming vs. traditional programming.... 15
- How does an event controlled application run?.... 16
- Three Programming Steps.... 16

Objects and classes.... 17
- Inheritance of classes.... 18

Statements and expressions.... 19
Multi-Line statements.... 19
Variables and Data Types.... 19
- Declaration of Variables.... 20
- Declarations of variables in different scopes.... 21
- Use of the 'Public'-Statement.... 22
- Use of the 'Private'-Statement.... 22

Local variables....23
Assignment-statement....23
Lifetime of variables....24
Place of declaration....24
Data Types....24
Simple Data Types....25
Data Type Size....25
Class types/Objects....26
Type NSObject....26
Comments....26
Way of naming....26
Literals....27
Expressions....27
Constants....28
Operators and Operands....29
Operators for calculating....29
Increment and Decrement....30
Comparison....30
Logical operators (Boolean Operators)....31
Other Operators....32
Operator order....32
Avoiding name collision....32
Editing source code....33
Working with objects....33
Create new objects....33
Use of Init functions....33
Create a class....34
Classes are not executed, but methods are....34
Accessing objects....35
Accessing instance-variables....35
Instance-methods....35
Calling methods....36
References to objects....36
Copying objects....36
Comparison of objects....36
Creating object variables....36
Declaration of an object variable....37
Assignment of objects to an object-variable....38
Use current instance or object / Me....38
Subclassing and inheritance....38
Hidden Methods....39
Overriding methods....39
Calling an overridden method....40
Hiding data....40
Scope modifiers....41
Arrays....42
Control of the program flow....42
Decisions....42
Single decision - If....43
Select....44

Loop-statements....45
For Next....46
For Each....47
Other kind of loops....47
Do While...Loop....48
Explicitly leave a loop....49
Explicitly test a condition....49
Nested control structures....50
Procedures / Methods....50
Sub-Procedure....51
Function-Procedure....51
Arguments....51
Named arguments....52
Writing function-procedure....52
Call of sub-procedures and function-procedures....53
Add comments to a procedure....53
Hints for calling procedures....53
Leave procedure....53
Use of return value after calling a function-procedure....53
Writing recursive procedures....54
Functions....55
Function Return....55
Modifiers / Scopes....55
Local scope....56
Classes....57
Classes unlike procedures are not executed....57
Edit Class....57
The Objective-Basic development environment....57
Windows....58
Toolbar....58
Editor....59
Classes and objects of Objective-Basic....59
Projects....59
Interfaces....59
Appendix....60
Application Bundles....66
PackageMaker....67
Copyright....67

About This Document

Welcome to this document, your guide for the development of your Objective-Basic applications. In this manual you find some necessary information to successfully build your Objective-Basic programs. If you complete this document, you will be able to write some simple Objective-Basic programs.

When you begin with Objective-Basic read the first chapters to learn about the programming language. Then try out the examples shipping with Objective-Basic and of this manual. The principal purpose of this manual is to give you an overview about the programming language at all. The less experienced Objective-Basic programmer receives an introduction to the programming language Objective-Basic. The object-oriented and other features of Objective-Basic are explained by some examples of the most important elements of Objective-Basic.

You will find a complete reference of the Cocoa frameworks and the Objective-Basic framework inside Objective-Basic's integrated development environment and therefore this manual comes without a detailed reference.

Why You Should Read It

This document is written both for people with some background knowledge and for absolute beginners. Experienced developers should feel very comfortable. This manual is meant for people, who:

- want to learn a simple and powerful programming language for Mac OS X;
- are experienced C/C++, Java, or BASIC developers, who want to switch to Objective-Basic or want to extend their abilities;
- or have heard of a new programming language for Mac OS X and want to learn it.

This document is an asset for anyone doing serious development with Objective-Basic; gives you a solid background in the fundamentals of Objective-Basic; and takes you step by step through the features of the language.

Note Of Thanks

We are very grateful to all the people who helped me to complete this manual.

About The Author

Bernd Noetscher is a software developer and the main developer of the Objective-Basic programming language.

Supporter

Thank you for proofreading this manual and for beta testing.

Thanks to everyone on the Internet who submitted corrections and suggestions; you have been tremendously helpful in improving the quality of this manual and I could not have done it without you: Thank you all!

Tell A Friend!

Like it? Love it?

Help us spread the word about Objective-Basic.

When you share Objective-Basic Software, you will make your friends happy and give us a hand in getting the word out around the world.

Objective-Basic Needs Your Help!

And the best way to help Objective-Basic (and to help Mac OS X to become an alternative to Windows) is to buy and use the Objective-Basic Professional Edition. With your help and financial support, it will be possible to continue the development of Objective-Basic for many

years.

You can help Mac OS X, when you help Objective-Basic, because Objective-Basic is an important piece of the puzzle showing the needed software for leaving Windows far behind. With your support it will be possible to show the world that BASIC development software do not have to be limited and difficult.

It is up to you!

Give Us Feedback

Please help us to improve this book. As the reader, you are the most important commentator and critic of this document. We respect your opinion and would like to know what improvements we could make. If you find an error in any example program or in the text, sent an e-mail to sales@objective-basic.com. Thank you!

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

Our primary goal is the satisfied customer. In order to improve, we need your help. If we have made any mistakes, please tell us. Please write to sales@objective-basic.com – we would like to get positive comments, too. Thank you very much.

New Set Of Manuals

Objective-Basic has aroused much Internet interest in the Internet among businesses and the developers. In response, we documented this exciting new technology with a set of new manuals. This series of manuals covers language references, introductory volumes, API references, and advanced topics of programming in Objective-Basic, such as databases and networks.

Installation

Simply open the disk image and copy the Objective-Basic application

where you like.

System Requirements are Mac OS X Lion Leopard (10.7 or higher).

Before you can use Objective-Basic, you must have installed the developer tools of Apple, which are on your Mac OS X installation disc or App store (Xcode and others).

Introduction

You heard probably already much about Objective-Basic and have probably downloaded Objective-Basic and played a little with the sample programs. And now you want to have your own Objective-Basic programs... Have you ever wanted to know how a computer program works? If so, there might be a computer programmer somewhere inside you, waiting to get out.

You might have found computer programming not only difficult but downright frightening. Are you going nearly crazy, if you try to write a simple script file? If you feel this way, this manual is here to prove that programming your computer is fun, rewarding, and not difficult. But before you get started with developing computer programs, you ought to have a basic understanding of what programming is all about. You probably have some ideas about what a program is and how it works. After reading this introduction, you may find that your knowledge about programming is good or you know... In either case, it is worth it...

Cocoa

Did you ever want to write programs that look and operate like other MacOS X applications? Hesitate to get hands on Xcode and learn Objective-C? But you have the experience of simplified object-oriented BASIC programming? Then, Objective-Basic seems to be the right decision. Objective-Basic is an Integrated Development Environment for using the Cocoa Framework with an object-oriented BASIC dialect.

Cocoa is an object-oriented application environment designed specifically for developing Mac OS X-only native applications quickly and efficiently. The Cocoa frameworks include a complete set of classes, and for developers starting new Mac OS X-only projects, Cocoa

provides the fastest way to full-featured, extensible, and maintainable applications. You can bring applications from UNIX and Windows to Mac OS X quickly by using Cocoa to build state-of-the-art Aqua user interfaces while retaining most existing core code.

Apple recommends that all new applications written for Mac OS X use Cocoa.

Cocoa is a collection of software objects that implement almost all features common to Mac OS X applications. Programmers extend the Cocoa objects to provide application-specific features. The Cocoa objects are reused in every Cocoa application so that programmers can concentrate on adding unique value with each line of code rather than constantly reimplementing common features or struggling to access operating system services. Significant applications can be built with very little code.

Cocoa is distinguished from other object-oriented development environments in several ways: Cocoa is mature, consistent, and broad. Cocoa is extraordinarily extensible, flexible and dynamic.

Because you use Cocoa and the default compiler of Apple (clang) through Objective-Basic the number of possible bugs, when you create your Mac application, is reduced nearly to zero. Objective-Basic is a mature tool, ready for production!

There are two ways to develop Cocoa applications with Objective-Basic. The first one is the difficult approach, by directly using the Cocoa functions. You are responsible for creating the right structure for your files and you must know many Cocoa classes in detail. The second and much easier way to build Cocoa apps is to use the Interface Builder comming with Mac OS X, enabling you to write Cocoa applications in a short amount of time, as it provides you with a set of files and structure.

In Objective-Basic memory management is automatically managed by a garbage collector provided by Cocoa.

What is a Computer Program?

A computer program is a list of instructions telling a computer what to

do. The computer follows these instructions or commands, one by one, until it reaches the end of the program.

Each line in a computer program is usually a single command that the computer must do. Each command does only a very small task, such as printing a name on the screen or adding four numbers. When you put hundreds, thousands, or even hundreds of thousands of these commands together, your computer can do great things: calculate mathematical problems very quickly, print a document, draw pictures, or play computer games.

As you can see in the next paragraph, computer programs can be written in one of many different programming languages.

Programming Languages

Computers do not understand German or English or any other human language. They cannot even understand BASIC, the computer language upon which Objective-Basic is based. Computers understand only one thing, machine language, which is entirely composed of the numbers 1 or 0. Programming languages like BASIC allow people to write programs in an English-like language. The BASIC interpreter changes the program into machine language so the computer can understand it. Objective-Basic programs are a dialect of the BASIC computer language that was developed not to help computers, but to help people make sense out of the numerical code machines use to function. Objective-Basic replaces many of the numbers used by machine language with words and symbols we can more easily understand and remember. Moreover, it enables a programmer to visually assemble a program's window from parts in a toolbox. It is much easier for you to communicate with your computer as an software developer; instead of thinking as a machine you can think as an human being.

History of the Name

Sometimes, you see the name of the BASIC language spelled with lowercase letters like this: Basic. Even Objective-Basic uses this spelling. However, BASIC actually started out as an acronym, which is why you also see the name spelled in all capital letters. The BASIC acronym stands for Beginner's All-purpose Symbolic Instruction Code. A

BASIC program uses symbols and words (and a few numbers) that people can understand. How it is possible that the computer can understand and run BASIC?

When you load Objective-Basic you are also loading a compiler. A compiler is a special program that takes the words and symbols from a Objective-Basic program and translates them into machine language the computer can understand. Your computer would not have any idea what to do with your program without the compiler's interpretation of your programs. Many computer languages exist, including Java, Objective-C and BASIC. However, all computer languages have one thing in common: they can be read by humans and therefore must be converted to machine language before the computer can understand them.

All Kinds of BASIC

There are many versions of the BASIC language, of which Objective-Basic is only one. All these software packages allow you to create computer programs with BASIC, but they all implement the BASIC language slightly differently.

Only Objective-Basic enables you to write Mac OS X modern Cocoa applications.

Compilers and Interpreters

Some computer languages, such as some types of BASIC, convert a program to machine language one line at a time as the program runs. Other languages, such as Java and Objective-Basic, use a compiler to convert the entire program all at once before the program runs. In any case, all programming languages must be converted to machine language in order for the computer to understand the program.

A compiler changes your program into an executable file that can be run directly, without a compiler or interpreter. An executable program is actually a machine language program that is ready for your computer to read and understand. With few exceptions, most computer programming languages have a compiler. It can be difficult to distinguish between compilers and interpreters, because a few programs, such as Java, are a mix of compiler and interpreter.

The Programming Process

Now that you know something about computer programs, how do you go about writing one? Creating a computer program is easy, though it can be a long process. Writing a Objective-Basic program requires development steps similar to those you use when writing a paper or report. The following list outlines these steps:

1. Come up with a program concept and sketch out on paper how it might look on the screen.
2. Create the program.
3. Save the program to disk.
4. Run the program and see how it works.
5. Fix programming errors.
6. Go back to step 3.

Most of the steps in the programming process are repeated over and over as errors are discovered and corrected. Even experienced programmers cannot write error-free programs unless the program is extremely short. Programmers often spend more time fine-tuning their programs than they do writing them initially. It is important to do the fine-tuning, because we are not as logical as we like to think. Moreover, human minds are incapable of remembering every detail required to make a program run perfectly. When a program crashes or does something unexpected we have to find errors hiding within code. Computer experts say there is no such thing as a program without bugs. After you start writing full-length programs, you will see how true this statement is.

Attack of the Bugs

Bugs are programming errors that stop your program from running correctly. Before a programmer can release his or her program to the public, he or she must correct as many errors as possible.

Is Programming difficult?

Well, yes and no.

It is easy to learn to write little programs with Objective-Basic. The language is logical, English-like, and easy to understand. With only

minimal practice, you can write many useful and fun programs. Actually, what you will learn in this book is enough programming for just about anyone who is not planning to be a professional programmer.

However, if you want to make programming a career, you have much to learn that is not covered in this book. For example, consider a word-processing program such as OpenOffice, which took dozens of programmers many years to write. To write such complex software, you must have good knowledge of how your computer works. Additionally, you must have spent many years learning the skills of professional computer programming.

Still, there is a lot you can do with Objective-Basic whether you are interested in writing utilities, small applications, or even computer games. And, step by step, you will discover programming in Objective-Basic is not as difficult as you might have thought.

How much do I need to know to use it?

It is possible to develop Objective-Basic programs without having to write any lines of source code. There are many places to obtain Objective-Basic programs, such as books, the Internet, and even your friends and colleagues. You can drop these programs into Objective-Basic and have an application ready to use. However, sometimes these programs need modifications and this is where real challenges to your programming expertise occurs.

Introduction to the Objective-Basic programming language

Welcome to the Objective-Basic programming language. This chapter is concerned with what exactly Objective-Basic is; why you should learn Objective-Basic and what makes Objective-Basic different from other programming languages; first steps inside Objective-Basic, which background knowledge is needed and some basic knowledge; and how to write your first Objective-Basic program.

What is Objective-Basic?

Objective-Basic is a powerful programming language designed to be intuitive and easy to learn. Objective-Basic is a new programming

language, a further BASIC dialect related to Visual Basic and Objective-C. More precisely, Objective-Basic is an object-oriented and event-controlled programming language and is designed particularly for the needs of GUI developers.

Objective-C developers feel that BASIC is a beginner language, but with Objective-Basic you can write many applications that may otherwise have been written in the more difficult Objective-C.

Briefly and well in a sentence

Objective-Basic is an easy to use, object-oriented, compiled, stable, independent, fast and modern programming language.

Objective-Basic's Past, Present and Future

At present, Objective-Basic is under construction. It will become more user-friendly, more efficient, and get more programming libraries and new object-oriented features.

What is still needed to know about Objective-Basic?

Objective-Basic is easy to learn: Objective-Basic is an easy to use language, designed for rapid development and easy error tracing. Objective-Basic is modeled after standard BASIC and Objective-C. It contains object-oriented ideas of Objective-C, whereby confusing elements were omitted. Objective-Basic is actually a complete and an elegant programming language.

Objective-Basic is object-oriented

That means for you as a programmer that you concentrate on the data and on the methods in your application - how to manipulate the data, instead of thinking only in procedures. If you get to know with it, how powerfully this new object-oriented paradigma is, you will get used to it, because you will see how easy it is to develop re-useable complex application programs clearly and in a modular way. Contrary to other programming languages Objective-Basic is designed from the beginning as an object-oriented programming language. So all things in Objective-Basic are objects; even the simple data types, like numeric and boolean values.

Stable

Objective-Basic is designed to write very reliable and durable software. Of course, Objective-Basic does not eliminate the need for quality control. It is still possible to write unreliable software. However Objective-Basic eliminates certain kinds of programming errors so that it is easier to write reliable software. The exception handling is a new possibility. An exception is a signal for a exceptional program condition, which occurs like an error. When you use the try/catch/finally statements, you can place all your error processing routines at one place, making it easier to manage and fix errors.

Objective-Basic is Fast

Objective-Basic is a compiled programming language. Therefore, it is fast as Objective-C or C. Objective-Basic programs are compiled into Objective-C code, which is itself compiled using Apple's default Objective-C compiler for Mac OS X.

Rapid application development

You can use Objective-Basic for visual programming features to quickly develop Objective-Basic applications. Actually, you use the default tool for GUI development on Mac: Xcode's Interface Builder developed by Apple.

The Xcode's Interface Builder let you point and click to:

1. Design the user interface for your program;
2. Specify the behavior of the user interface elements; and
3. Define the relationship between the user interface and the rest of your program

In addition to its visual programming features, Objective-Basic gives you the ability to quickly complete many tasks, including create new program elements. In Objective- Basic, a program element is one of the following:

Project: the top-level program element in Objective-Basic. A project contains classes and definitions. Currently, the project management is directory based, which means a project is defined through the context of a directory. Inside this directory, there is a directory containing the

executable with resources like images. Additional, inside your project directory are the source codes files in Objective-Basic and the automatically generated Objective-C files.

Interface definitions (xib/nib files): the Objective-Basic visual programming feature.

Class: the Objective-Basic language construct. Classes contain the source codes.

How Can I Get Objective-Basic?

You can buy Objective-Basic Professional. You can get it directly from its maker www.objective-basic.com, where you can purchase a commercial license. The buyer receives a license, for Mac OS X and the right to write commercial programs without paying fees. If you have questions, send an e-mail to: sales@objective-Basic.com.

On which platform is Objective-Basic available?

At this time, versions of Objective-Basic exists for Mac OS X only.

Where can I find more information about Objective-Basic? Examples, documentation, and news?

First read this book. You can also get news and new information on http://www.objective-basic.com/. Many programming examples are posted on the Internet.

Software development with Objective-Basic

A typical Objective-Basic application consists of interface elements, classes, and other objects, which together are your application. Interface elements have events and generates events. You can react by assigning Objective-Basic-code to such events.

Event controlled programming vs. traditional programming

When a traditional, procedure-built program runs, the application controls the executable parts of the program, ignoring the events. The application starts with the first code of line and goes through its

source codes as the developer has defined. If needed some procedures are called.

In event controlled applications, user action or a system event triggers the execution of the event procedure. The order in which the code is executed depends upon the order of the events, which occur based upon user actions. This is the principle of graphical user interfaces and event controlled programming. The user performs an action and your program reacts.

How does an event controlled application run?

An event is an action recognized by your windows or interface controls. Objects in Objective-Basic know some predefined events and react to them automatically. To get a control to react to an event, write an event procedure for this event. A typical application runs as follows:

1. The user starts the application.
2. The window or control receives an event. The event can be triggered by a user-action (e.g. key pressed) or by your code (e.g. event 'Open' if your code opens a form).
3. If there is a event-procedure for the event, the desired event code executes
4. The application waits for the next event

Three Programming Steps

To create a simple Objective-Basic program, you need only complete three steps, after which you will have a program that can be run outside of Objective-Basic's programming environment just like any other application you may have installed on your computer. The three steps are as follows:

1. Create the program's user interface
2. Write the program source code, which makes the program do what it is supposed to do
3. Compile the program into an executable file to run as a standalone application
4.

Of course, there are many details involved in each of these three

steps, especially if you are writing a lengthy program. As you work, complete these steps in the order listed above. You should frequently alternate between steps 2 and 1 to fine-tune your user interface. You might even return to step 2 from step 3 as you discover problems after compiling your program.

Objects and classes

Objective-Basic is an object-oriented programming language. Object-oriented, in Objective-Basic terms, contains the following ideas:

- Classes and objects in Objective-Basic
- Creating objects
- Garbage collection to release unused objects

There are no class variables or class functions yet, all variables and functions are useable together with objects only due to simplify the object-orientation. Though class variables and functions might be added in future releases. Though all functions and variables can be used with the singleton pattern, which in the end acts like a class variables and class functions.

Additional concepts are:
- The extension of classes in order to create a sub-class
- The override of class methods and polymorphism when it comes to calling methods

If you are a Objective-C-developer or a experienced developer in another object-oriented programming language, you will know many of the concepts of Objective-Basic.

Object-oriented programming means every object is part of another object.

For example, in the real world, cars consist of many objects like windows, steering wheels, and so on. There is only one class (building plan) for a window or steering wheel. Therefore there exists a class window and a class steering wheel, but many objects window or

steering wheel, which are the instances of a class. An instance of a class is also called an object of a class. Classes exist when you write your code. Objects are created when Objective-Basic runs your code using your classes!

Every class has the following elements:

1. Super-class
2. Attributes (variables, constants...)
3. Methods/Procedures

Super-class means all attributes and methods of the parent-class apply to the new class in addition to attributes and methods you declare in the new class. Attributes can be constants, or variables containing to a class.

Methods are procedures and functions of a class. Methods are arguably the most important part of any object-oriented programming language. Whereas classes and objects provide the framework and class and instance variables provide a way of holding that class' or object's attributes, the methods actually provide an object's behavior and define how the object interacts with other objects in the system. Variables and methods can be related to a class (one time in memory) or can be instance-related (as many as instances of that class exist) in the declaration of a class (outside a method):

Instance-variable, e.g. Dim instanceVar
Instance-method, e.g. Sub instanceSub()
Instance-method or instance-variable can only be used together with an object. Instance-variables are like local variables of a procedure. Local variables only exist after calling the procedure, instance-variables only exist within the lifetime of an object. Classes can give all its elements to its child, which is called inheritance.

Inheritance of classes

Inheritance is an important part of object-oriented programming. Objective-Basic, like Objective-C, supports single inheritance, meaning every class inherits one parent class. It is not possible to inherit from many classes. In the far future, it will be possible to use an interface to do many things you could do with multi-inheritance. Unlike in

Objective-C, all objects in Objective-Basic are created when you declare them with a "Dim NAME As TYPE" statement. You do not need to use "alloc" or "init" like in Objective-C. For details and exceptions read the language reference.

A class is a collection of data definitions and methods working on those data. Data definitions and methods describe the content and the behavior of an object. Objects are instances of a class. You cannot do much with a single class but can do much more with an instance of a class, a single object containing variables and methods. Object-oriented means the objects are the centerpiece, not methods and variables.

Statements and expressions

Statements are a complete action in Objective-Basic. Statements can contain keywords, operators, variables, constants, and expressions. Every statement can be categorized by one of the following:

- Statements as declaration or definition for class, variable, constant, procedure, function, method
- Assignment statements, assigning a value or expression to a variable or constant
- Executable statements performing an action. These statements can execute a method, function, procedure, loop, or condition
- Executable statements often contain conditional or mathematical operators (even complex expressions).

Multi-Line statements

Use _ to connect to create a multi-line statement:

e.g.

```
Dim m As NSMenu _

= mRecentProjects.Submenu
```

Variables and Data Types

While computing expressions you may need to store values for a short

time. For example, you would like to calculate different values, compare these values, and depending on those values, perform different operations. In order to compare values you need to store them. Storing does not save the values on disk but in memory because you need the values only when running your program.

Objective-Basic uses variables to store values at runtime of your program. After stopping your program, all variables (even objects) are deleted. A variable can change its value at any time. You can change its value by assigning an expression, another variable, or a constant to it. A variable is a space in memory used by Objective-Basic to store values so variables are not stored in a file. Like a file, a variable has a name (used to access the variable and to identify it) and also a data type to specify the type of information a variable holds.

Objective-Basic knows instance-variables and local variables. Instance-variables are part of an object and local variables are part of a procedure, function, or sub (also called method).

Declaration of Variables

Before using variables, you must declare them. You must define the name and the data type of a variable.

The ‘Dim’-statement declares a variable. ‘Dim’-statements are one of many variable declaration statements, which slightly differs from the others. You may combine 'Dim' together with ‘Public’ or ‘Private,’.

```
Dim myName As String
```

Variables can be declared everywhere in procedures, functions, and subs, not only at the top of procedures, functions, and subs. However, most or the time they are at the top. Every line of your source code may contain only one declaration statements for one variable. Which data type the variables have, depends upon the way you declare the variables. The default data type is ‘id’.

```
Dim name
Dim sirname As String
Dim lastname As String
```

name is type ‘id’ sirname is type ‘String’ lastname is type ‘String’

Or explicitly with different types:

To use a data type you have to write it for every variable, otherwise the default data type would be used.

You may assign a value to the variable directly in the same line of declaration.

```
Dim name As String = "sunny sun"
```

Names of variables must not be longer than 128 characters and must not use all characters possible. Do not use periods, commas and other non-writable characters. Only the underscore (_) is allowed. Important! Objective-Basic does differentiate between a variable name written in lowercase or uppercase. In fact you cannot write it differently in every line. Always write your variable names consistently. So Named, named, naMED are different variables.

Use the following rules to name functions, subs, constants, variables, or arguments in Objective-Basic: Use a letter (A-Z, a- z) or underscore (_) as your first character

Never use whitespace ( ), period (.), exclamation mark (!), or the following characters : @, &, $, #, „ in your names.

The name may contain numbers but must not start with a number.

Do not use names already in use by predefined elements of the Objective-Basic language, such as keywords (e.g. ‘If’), built-ins, classes etc. This would lead to a name collision in which the original language element would be expected. Your program will not compile.

You cannot use variables with the same name in the same scope. e.g. you cannot declare variable ‘age’ two times in a function, but you can declare ‘age’ in another function (at different places).

Declarations of variables in different scopes

You can place a declaration statement inside a function, sub, or method to declare a variable in local scope. Additionally, you can

place a declaration to declare an instance-variable in the declaration section of a class. The place of the declaration determines the scope of the variable. The scope of a variable cannot change at runtime of your program, but you can have different variables with the same name at different places. For example, if you have the declaration of a variable named ‘price’ in a function, and the same declaration in a class, all uses of this variable in the function are related to the local variable ‘price,’ all uses outside the procedure are related to the instance variable ‘price.’

A variable named sName with type of ‘String’ is declared in the following example:

```
Dim sName As String
```

If this statement is part of a function, then it is possible to use this variable in the same function only in the lines after the declaration. Is this statement part of a class declaration section, you can use it in all methods of the class, but not outside the class (in fact it is declared as ‘Private’ implicitly). To be able to use it everywhere, you can use the keyword ‘Public’.

Example:

```
Public Dim sName As String
```

Use of the 'Public'-Statement

You can use the ‘Public’-statement to declare public variables in class scope, making the variable accessible from everywhere.

```
Public Dim sName As String
```

Use of the 'Private'-Statement

Use the ‘Private’-statement to declare private variables in class scope, making the variable accessible only from the same scope (class scope, all class methods).

```
Private Dim myName As String
```

If the ‘Dim’-statement is used in class scope without the keyword 'Private', it is treated as a ‘Private’-statement. Use the ‘Private’-Statement to have cleaner, more readable code.

Local variables

The value of a local variable is available inside the procedure only. You cannot access the value from outside the procedure. This makes it possible to have the same variable name for different variables in different procedures without name collision. Example:

```
Sub test1()
  Dim i As Integer
  i = 1
End Sub

Sub test2()
  Dim i As Integer
  i = 19
End Sub
```

Assignment-statement

Assignment-statements give or assign a variable a value or expression with variables, constants, numbers, etc. An assignment always includes a sign. The following example shows an assignment.

```
Dim yourName As String
yourName = "Nadja"
```

After the declaration of a variable, you can use it, e.g. make an assignment.

```
Dim Cool As Boolean
Dim myName As String
Cool = Yes
Name = "Julie"
```

Cool contains ‘Yes’, name contains ‘Julie’ after the assignment.

Lifetime of variables

The lifetime of variables is the time in which the variable exists or has a value. The value of a variable can be changed during its lifetime. If a variable is outside its scope, it has no value. If a procedure is executed, a variable exists after its ‘Dim’-statement. Local variables exists until the procedure completed; after reentering the procedure all variables are created again after their respective ‘Dim’-statements.

So if you do not change the value of a variable during its lifetime in a program, it contains the initialized value at program start. A variable declared in a sub contains an assigned value until the sub is left. If the same sub is called again, the value is reset to the initialized value.

If you use instance-variables you must consider that they only exist together with the instance (object) to which they belong.

Place of declaration

Declare every variable and every procedure inside a class.

Syntax:

```
Dim Name [As Type] [= Expression]

[Public | Private] Dim Name [As Type] [= Expression]
```

Data Types

Data types describe the type of data stored inside a variable. If you declare a variable you must also give it a data type. Objective-Basic supports most Objective-C data types . You can use one of the following data types:

- One of the simple data types (e.g. ‘Integer’)
- Name of a Objective-Basic Framework class
- Name of builtin class (Cocoa class)
- User defined class

Simple Data Types

Simple data types store numbers and text. Simple data types are simple because they are built in within Objective-Basic and are not complex like classes. Every simple data type has borders determining the size of the stored number. If a value is too big it loses precision, meaning it loses information!

The following table contains all Objective-Basic supported datatypes with their needed space in memory.

Simple data types are internally stored as objects in Objective-Basic in order to get some extra functionality like runtime-checking.

Data Type Size

Boolean: 'Yes' or 'No'(NSNumber object)

```
Values can only be 'Yes' or 'No'.
```

Integer: -2^31 till +2^31 –1. (internally uses NSInteger, size depends on 32/64-bit CPU)

```
Values can be -2^31 >= and    +2^32. You can use integer variables to
simulate enumerations, such as 0 = black, 1 = white and so on.
```

OR

```
Stored as 64-bit numbers. Values can be -2^63 >= and    +2^64.
```

String:

```
Actually, it is a class (NSMutableString) of the Cocoa Framework.
```

Single (NSNumber object):

```
Sored as floating numbers with single precision. Values can be between
-3.402823^38 or -1.401298^-45 or between 1.401298^-45 until 3.402823^38.
```

Double (NSNumber object):

```
They are stored as floating numbers with double precision. Values can be
between -1.79769313486232^308 bis -4.94065645841247^-324 or between
4.94065645841247^-324 till 1.79769313486232^308
```

NSObject

```
Stored as 32-bit or 64-bit references (4 or 8 Bytes).
```

Class types/Objects

Variables can store objects (actually, references to objects). These variables are object variables of data type ‘NSObject’.

Though an object variable can contain any object (actually, the reference to that object), the binding is done at runtime (late binding).

Type NSObject

The data type ‘id’ is automatically used if you do not specifiy a data type for an argument, constant, procedure or variable.

The following example creates two variables of type ‘id’:

```
Dim myVar
Dim yourVar As id
```

Additionally, a ‘id’ can store the following value: Nil

Comments

The comment symbol (’) is used in many code lines in this book. Comments can explain a procedure or a statement. Objective-Basic ignores all comments while compiling and running your program. To write a comment, use the symbol (’) followed by the text of the comment. Comments are printed in green on screen. Objective-Basic recognizes comments, as shown below.

```
' this is a comment till the end of this line
```

Comments are extremely helpful when it comes to explaining your code to other programmers. So comments, normally, describe how your program works.

Way of naming

When coding in Objective-Basic, declare and name elements, like procedures (functions and subs), variables, and constants and so on. All names

- must start with a letter;
- must contain letters, numbers, or the sign (_); periods and commas are forbidden;
- must not be longer than a defined length; and
- must not be equal to reserved words

A reserved word is a part of Objective-Basic and has a predefined meaning. These include keywords (e.g. ‘If’ or ‘Then’), built-in-functions and operators (e.g. ‘Mod’). A complete list of reserved words is available in the reference of Objective-Basic. Be aware that names are case sensitive, which means you must it write with the same lowercase or uppercase letters everywhere.

Literals

Besides keywords, symbols, and names, a Objective-Basic program contains literals. A literal is a number or string representing a value. There are different numerical literals.

Byte, Short, Integer, Long -1, 2, -44, 4453
Hex - 0xAA43
Single (Decimal), always English formatted - 21.32, 0.3F, -435.235421.21
Double (Decimal), always English formatted - 212.23, 12E1F
Boolean - When you cast a numerical value to Boolean. Values with 0 are ‘No’. Other values are ‘Yes’.
String - Is simply text, but it must start with a (") and end with a (") so that Objective-Basic can recognize it as string. A (\") inside a string is interpreted as single ("). In fact, all Objective-C escape codes are applied. So you may use (\n) to insert a return in a string for example.

Expressions

Expressions represent values. They can contain keywords, operator, variables, constants or even other expressions. Examples for expressions are:

- 1 + 9 (number operator number)
- myVar (variable)

Expressions can return a value or not. If an expressions is to the right of an assignment statement (, the expression gives or must give a value back.

```
myVar = 1 + 9
```

1 + 9 is 10 and this expression value (10) is stored in myVar. It is exactly the same when the expression is used as a parameter in a function call.

```
MyProcedure(1 + 9)
```

Expressions are calculated at runtime and represents a value. The result can be assigned to a variable or to other expressions listed above.

Constants

Constants are similar to variables but they cannot change values. When you declare a constant you assign a value to it that cannot be altered during lifetime of your program. Example:

```
Const border = 377
```

This ‘Const’-Statement declares a constant and a value of 377.

Use the same rules for declaring constants as for variables. A ‘Const’-statement has the same scope as the variable statement. To declare a constant inside a procedure, place the ‘Const’-statement inside this procedure (normally one of the first statements). To declare a constant as accessible for all procedures, place it within the class where the procedures are.

You may change the visibility of a class const to other class by using 'Public' or 'Private'.

```
Public Const a = 34
Private Const b = "Hey"
```

This following ‘Const’-Statement declares a constant conAge as public

with data type ‘Integer’ and a value of 34.

```
Public Const conAge = 34
```

You must not declare many constants in one line.

```
Const Name = Expression
Const Name = Expression
[Public | Private] Const Name = Expression
```

.

Operators and Operands

Objective-Basic supports most Objective-C operators. Operators are ‘+’ or ‘-’ for example. Operands are numbers, strings, or variables (or expressions). There are different kind of operators.

Operators for calculating

They are used for calculating of two operands. Every of those operators needs two operators. The minus (-) can be used to negate values, like -9 means negative 9, +9 means positive 9, 9 means positive 9

Calculating operators are:

```
+ Addition – Addition adds two numbers. If you add a number and a
string
```

Objective-Basic cannot be sure if the result should be a number or string. In this case, you must use the ‘&’ Operator instead of ‘+’ for strings. e.g. “test“ + 4

- - Subtraction `– Subtraction subtracts a number from another number. e.g. 5 – 7`
- * Multiplication - Use to multiply two numbers e.g. 33 * 7
- / Division - Use to divide two numbers. The result is floating point. e.g. 2 / 5
- \ Integer-Division - The result has data type ‘Integer’. e.g. 29 / 5

- Mod Modulus, also known as remainder of Integer-Division – Returns the remainder of a result of an Integer-Division e.g. 45 Mod 10

Integer-Division results in an 'Integer' value.

Example: x is 10, y is 4

```
Log(x + y) ' is 14
Log(x – y) ' is 6
Log(x * y) ' is 40
Log(x / y) ' is 2.5
Log(x \ y) ' is 2
Log(x Mod y) ' is 5
```

Normally, the result has the data type of the operand (= expression) with more precision.

Increment and Decrement

To increment or decrement, use var = var + 1 or var = var – 1.

Comparison

Objective-Basic has different operators to compare expressions: These operators compare between two operands (expressions) and the result is 'Yes' or 'No':

```
= Equal
```

e.g. x = 3

```
<> Unequal
```

e.g. x <> y

```
< Smaller
```

e.g. x < 3

```
> Bigger
```

e.g. x > 4

```
<= Smaller or equal
```

e.g. x <= 4

```
>= Bigger or equal
```

e.g. x >= 3

Logical operators (Boolean Operators)

Use logical operators for performing bit operations or combining bits together. The result is ‘Yes’ or ‘No’. Objective-Basic supports true logical operators for decisions (’AndAlso’, ‘OrElse’).

```
AndAlso
```

Both operands (expressions) must be ‘True’

```
OrElse - One of the operands (expressions) must be ‘True’
```

You can also use bit-operators instead of ‘AndAlso’ or ‘OrElse’. Bit Operators

- And and - e.g. 4 And 6 - Useful for doing logical conjunctions of two expressions. The result is ‘True’ if both expressions are ‘True.’
- Or or - e.g. 33 Or 8 - Useful for doing logical disjunctions of two expressions. The result is ‘True’ if one of both expressions is ‘True.’
- Xor exclusive or - e.g. 77 Xor 3 - The result is ‘False’ if both expressions are ‘True.’ Or the result is ‘True’ if one of both expressions is ‘True.’
- Not not - e.g. Not 5 -Useful if you would like to negate expressions.

Furthermore, ‘Not’ inverts a bit of a byte.

A result is 0 when all bits are not set (binary 0000 0000). You can use shift operators Shr or Shl.

Other Operators

- `^ Power IS NOT SUPPORTED THIS WAY – See the math function in the language reference to get a workaroud. Useful if you want to use power. If they are many powers in one expressions they are processed from left to right.`
- `. dot operator - Needed for calling methods`

Operator order

Objective-Basic supports the operator order of VB. Normally, an expression is executed from left to right following standard mathematical rules.

e.g: x = 10 + 10 / 2 results in 15 and not 10, because / is calculated before +.

Here is the overview about operator order/priority from top to bottom, which means '*' is executed before 'And.':

```
1. . ( )
2. Not, !, (unary +), (unary -)
3. ^
4. * / Mod \
5. & + -
6. < > < = >= = == ===
7. And
8. Or Xor
9. AndAlso
10.OrElse
```

Use paranthesis () to change the order.

e.g. X = (10 + 10) / 2 results in 10 and not 15, because + is calculated before / thanks to the braces.

Avoiding name collision

A name collision occurs when you use a name that was already defined in Objective- Basic (or even by Objective-Basic itself = keywords). Avoid name collisions by knowing the concept of scopes. There are different types of scopes in Objective-Basic: procedure scope, class

scope and the scope modifiers (’Public’, ‘Private’). You can have name collisions in the following situations:

• If an identifer can be accessed from different scopes
• If an identifier has different meanings in the same scope

Most name collisions can be avoided by using full qualified names of identifiers.

Editing source code

Editing source code is no different than editing text in a word processor. You have a cursor showing the current position you can type in. The find and replace commands within the editor work just as in your word processor, as well as many other common commands.

Working with objects

Because Objective-Basic is an object-oriented programming language, you probably would like to know how to use these features. The following issues need to be discussed:

How can I create a class?
How can I create an object (instance) of a class?
How can I use class-variables, class-methods, instance-variables and instance-methods?
How can I convert an object into another object?

Create new objects

Either you create an object based on a built-in class of Objective-Basic (Cocoa or Objective-Basic Framework), or you create it from your own defined class.

Use of Init functions

A variable, which contains the object and is declared, does create the object at once by calling its default init function. This variable only has the data type of an object and represents only a reference to an object just created. For each data type exists at least one init function, which has the same name as the object has.

```
Dim a As Array
```

Creates an object of array and assigns a reference to the object variable a.

```
Dim a As Array = Array(1, 2, 3, 4, 77)
```

Creates an object of array and assigns a reference to the object variable a. The array object contains the values 1, 2, 3, 4, and 77 as NSNumber objects.

Create a class

Normally every class is a child of the special class 'NSObject' directly or indirectly through another class. A class can inherit from another class (be child of that class). The declaration of a class consists of a class name and the super (parent) class name. Furthermore, you have init functions, sometimes a destructor (finalize method), some variables (instance-variables), and procedures (methods, instance-methods) and, of course, some constants.

Parts are:

- Variables
- Constants
- Functions
- Subs
- Events/Delegates/IBOutlets/IBActions/Actions
-

A class can only be declared inside a class file. It is not possible to declare many classes in one class-file.

Classes are not executed, but methods are

A new created class only contains the declaration part (class name and super class name), but no methods. These must be created by the programmer. Objective-Basic does not run any class, but the methods (functions and subs) inside the class.

Accessing objects

Accessing objects is done by using the dot operator (.).

Example:

objectInstanceName.constantname
objectInstanceName.variablename
objectInstanceName.subname
objectInstanceName.functionname

Accessing instance-variables

Accessing variables is done by using the dot operator (.).

Example:

```
myObject.variable
```

When accessing instance variables, use the name of the object variable:

```
objectName.instanceVariable
```

Please remember! Instance-variable are part of an object of a class, so a instance-variable exists how often objects exists of that class and only during the lifetime of the object.

Example:

```
Private Dim myInstanceVariable As String
```

Another example:

Accessing inside a method:

```
Dim o As myClass
o.instanceVariable = 99
```

Instance-methods

Instance-methods are only present when its object is present. You may use ‘Me’ inside instance-methods.

The method is by default an instance-method.

Calling methods

Again, use the dot opertator (.) to access the methods.

```
myObject.myInstanceMethod()

Sub myInstanceMethod
  ...
End Sub
```

References to objects

Using the '='-Assignment you can assign a reference of an object variable to another object variable. Both variables then reference the same object. If you would like to assign values, you must use 'Copy' to assign values.

Copying objects

In future version, if you would like to copy (or values), you must use 'Copy'. Meanwhile copying have to be done manually.

Comparison of objects

The '='-comparison operator tests if two object variables have the same value.

The '=='-comparison operator tests if two object variables points to the same object, ignoring whether both variables' objects might have the same content.

The '==='-comparison operator tests for equal classes.

Creating object variables

Think of an object as a variable, like it would be the object itself. You can access the object through a object variable, or call a object method.

How to create an object variable?

1. Declare an object variable 2. Assign an value to the object variable or assign an existing object.

Declaration of an object variable

To declare an object variable, use the ‘Dim’-statement or another declaration statement (’Public’, ‘Private’). A variable, which references an object, must be of type ‘NSObject’, or specific object type (like ‘String’...).

Some declaration of objects examples:

```
' Object1 as id (as general Object)
Dim Object1

' Object1
Dim Object1 As NSObject

' Object1 as String-Object of the Objective-Basic Framework
Dim Object1 As String
```

When using an object variable without a declaration, the object variable has the default data type ‘id’, which can be thought as an alias for 'NSObject'.

The declaration of an object variable with data type object is useful when you do not know which object type you would like to have in a generic sub. If you know the specific object, declare the object variable with the same data type as the object.

Example with normal object type and specific object type:

```
Dim Objekt1 As NSObject '  NSObject
Dim Objekt1 As Sample ' Sample
```

The declaration of specific object types has some advantages, such as type checking and more readable or faster code execution.

Because objects are not passed by values in procedures, an assignment of an object does not copy an object. Instead you must provide proper

methods in the classes that can copy an object.

Assignment of objects to an object-variable

Using the '='-statement lets you assign objects to an object variable. An object variable can have the value of 'Nil' or an object reference. Some examples:

```
Object1 = Object2 ' object reference

Object1 = Nil ' set no object reference

Dim myString As String
Dim myString2 As String

myString = myString2
```

Use current instance or object / Me

The keyword 'Me' references the current instance (or object) in which the code is currently executed.

Example:

```
Sub changeObjectColor(Object1 As myObject)
  Object1.myBackColor = myRGB(256, 256, 256)
End Sub

Sub otherMethod()
    changeObjectColor(Me) ' statement inside the object
End Sub
```

The keyword 'Super' for accessing the super class of the current object is not supported yet.

Subclassing and inheritance

There is another possibility to extend your own classes or Cocoa classes. This is called inheritance, meaning you use a super class as a

template for a new class in which you add new methods, variables, etc. All elements, like methods and variables of the super (parent) class, are automatically inside the new class (hidden) and can be accessed and used depending upon the declaration in the parent class. To change the behavior, override a method of the parent class (have the same name and arguments), so that the algorithm of the parent class uses the new method in the new class instead of the original method in the parent class. Inheritance is a very powerful and an important feature of object-oriented programming languages.

Using the file extension tells Objective-Basic which super class it uses for the new class.

Because, every class has a super class, all classes forms together a class hierarchy (or inheritance hierarchy). The inheritance hierarchy like a tree, with a root (parent class 'Object') and many twigs (many child classes). A famous example is the drawRect method of a View, which is considered to be overridden by you to implement custom drawing for your control. This is done when you would like to have a custom control.

Hidden Methods

An existing method, like a variable, in a parent class can be overloaded by a method in the child class through inheritance, though it is possible to access both methods with the same name. This is called hiding, because it is not automatically visible anymore. Hidden methods are also called overridden methods. Overriding takes place if the child class has a method with the same methods' signature (name, return type, same arguments) as the parent class. The parent method is now overridden.

In the future, not implemented yet: If you would like to access the parent method, enter 'Super' dot (.) and the method name:

```
Super.myMethod() ' access parent method myMethod() ' access me method
```

Overriding methods

Overriding occurs if the child class has a method with the same

method signature (name, return type, same arguments) as the parent class. The parent method is now overridden. If this method is called within the child class, the new defined method is used, even by the parent methods (included by the child class). Overriding methods is a very important technique in object-oriented programming. However, do not confuse overriding with overloading. Overloading means having many methods with the same name in a class but with different method signatures (different return type or arguments).

Objective-Basic treats variables and methods equally within a class. It is no different when it comes to overriding methods and hiding variables.

In the future, not implemented yet:

You can easily access hidden variables by casting its type to the parent type or using the keyword ‘Super’. The difference is an internal meaning because variables are different when used in child and parent class (they actually use their own variable), while methods are used by both (both parent and child class use the same method!).

Calling an overridden method

In the future, not implemented yet:

Access overridden methods by casting its object variable to the parent type or use the keyword ‘Super’.

If you override a method, you have two methods: the old method in the parent class and the new method in the child class. Access the old method explicitly using ‘Super.’ Access the new method using ‘Me’ or the standard call of a method.

Hiding data

Hiding data (variables and constants...) is a very important technique in object-oriented programming. Hiding data means not all data in a class can be accessed by other classes, but through methods of the same class in which the data is present so that it is only visible within

this class. This helps reduce mistakes when using internal data of a class by other classes or modules. You have public methods and variables that are accessible by all others and you have private methods and variables that should only be used by the class itself and those public methods. You automatically hiding data, when you use the ‘Private’ modifiers for variables and sometimes for methods. It depends on the code you write. Further reasons for hiding data are:

Internal variables, which are externally visible, mixing up the API for other coders etc. Hiding them leads to small, smart classes with a few variables and methods visible to others.

If a variable is visible you must document it. Hiding variables reduces documentation time.

Scope modifiers

To hide variables or methods you must declare them as ‘Private’:

```
Private Dim wings As Integer ' only visible within this class

Private Function countWings() ' only visible within this class
  ...
End Function
```

A private variable is visible for the class in which it has been declared. This means the methods of this class can only use this private variable. Private methods and variables are not visible to child classes of the class in which they are declared. Non-private methods and variables are always visible to child classes.

Besides ‘Private,’ there exists ‘Public’ as scope modifiers. Public means that every part of your program (any other class) can access and use the public declared part of your class (variable or method). The default modifier is ‘Private.’ If you do not write ‘Public’ in front of your variable or method, it is automatically declared as private.

Some tips for using scope modifiers. Use ‘Public’ for methods and variables that are used and accessible from everywhere.

Arrays

If you have worked with other programming languages, you understand the concept of arrays. Arrays are variables that look like a chain. All elements of that chain may have the same data type and can be accessed by an index. These variables have one name but a different index. You can use the array as one block or as one element of the array. Arrays are declared using the Cocoa NSMutableArray class.

```
Dim a As Array
a = Array("one", "two", three") ' declares an array with three strings
```

Using arrays leads to cleaner, more flexible code because you can use loops to use or access thousands of variables (the arrays). Arrays have an upper and a lower boundary and elements of that array are between those boundaries. The important role of new arrays is rising, because the more modern style of having classes imitating arrays with more flexibility and functionality, especially when using Cocoa, leads to cleaner code.

Objective-Basic supports arrays with unlimited capacity, arrays are based on Cocoa arrays which support one dimension, but multi-dimensions can be simulated by using the formula:

index = dim1 * dim2_size + dim2...

There are the Objective-Basic Framework classes ‘Array’, ‘Dictionary’ and ‘Data’, which should fit in most situations.

Control of the program flow

The statements that make the decision and loops are known as control structures.

Decisions

The term ‘decisions’ refers to the use of conditional statements to decide what to execute in your program. Conditional statements test if a given expression is ‘Yes’ or ‘No.’ Then, statements are executed. Normally a condition uses an expression in which a comparison operator is used to compare two values or variables.

Single decision - If

A single decision is used to execute a set of statements if a condition is set (’If’-statement). If the condition is ‘Yes’ then the statements after the ‘Then’ are executed and the statements after the ‘Else’ are skipped. If the condition is ‘No’, the statements after the ‘Else’ are executed. If the item after ‘Then’ is a line number, a goto is executed. If the condition is ‘Yes’ and there is no ‘Then’ statement on the same line, statements are executed until a line with an ‘End If’ is found.

Syntax:

```
If Expression Then
[Statements]
End If

If Expression Then
  [Statements]
Else
  [Statements]
End If

If Expression Then
  [Statements]
Else If Expression
  [Statements]
Else
  [Statements]
End If

If Expression Then
  [Statements]
Else If Expression
  [Statements]
Else If Expression
  [Statements]
Else
  [Statements]
End If

If Expression Then
  [Statements]
```

```
Else If Expression
  [Statements]
End If
```

‘Else If’ introduces a secondary condition in a ‘If’-statement.

‘Else’ introduces a default condition in a ‘If’-statement.

‘End If’ ends an ‘If’-statement.

‘If’ evaluates ‘expression’ and performs the ‘Then’-statement if it is ‘Yes’ or (optionally) the ‘Else’-statement if it is ‘No’. Objective-Basic only allows multi-line ‘If’-statements with ‘Else If’ and ‘Else’ cases, ending with ‘End If’. This works as zero is interpreted to be ‘No’ and any non-zero is interpreted to be ‘Yes’

Example:

```
Dim i As Integer
Dim n As Integer
If i 1 Then
  n = 11111
Else If i = 2 * 10 Then
  n = 22222
Else
  n = 33333
End If
```

Select

The ‘Select’-statement is much more complicated than the ‘If’-statement. In some situations, you may want to compare the same variable or expression with many different values and execute a different piece of code depending on which value it equals to. This is exactly what the ‘Select’-statement is for. ‘Select’ introduces a multi-line conditional selection statement.

The expression given as the argument to the ‘Select’-statement will be evaluated by ‘Case’-statements following. The ‘Select’-statement concludes with an ‘End Select’-statement. As currently implemented, ‘Case’-statements may be followed by string values, in this case

complex expression can be performed. Strings are compared case sensitive. The test expression can be ‘Yes,’ ‘No,’ a numeric, or string expression.

‘Select’ executes the statements after the ‘Case’-statement matching the ‘Select Case’-expression, then skips to the ‘End Select’-statement. If there is no match and a ‘Case Else’-statement is present, then execution defaults to the statements following the ‘Case Else.’ You may also use ‘Select’ together with the Objective-Basic Framework class ‘String’. It automatically generates the right Cocoa calls for you. See the Objective-Basic Framework ‘String’ examples for help.

Syntax:

```
Select Expression
Case Expression
  [Statements]
Case Expression
  [Statements]
Case Else
  [Statements]
End Select
```

Example:

```
Dim i As Double
Dim n As Integer
i = 4
Select i
Case 0
  n = 0
Case Else
  n = 999999
End Select
```

Loop-statements

The statements that control decisions and loops in Objective-Basic are called control structures. Normally, every command is executed only one time but in many cases it may be useful to run a command several times until a defined state has been reached. Loops repeat commands

depending upon a condition. Some loops repeat commands while a condition is ‘Yes,’ other loops repeat commands while a condition is ‘No.’ There are other loops repeating a fixed number of times and some repeat for all elements of a collection.

For Next

The For-Next loop is useful when you know how often statements should be repeated. For-Next defines a loop that runs a specified number of times.

Syntax:

```
For counter = start To stop [Step stepExpression]
  [Statements]
Next
```

‘Counter’ is a numerical variable used to store the current counter number while the loop is running. ‘Start’ is a numerical expression that determines the starting value of the loop counting. ‘Stop’ is a numerical expression that determines the ending value at which the loop will stop. To count in steps other than the default value of 1, use ‘stepExpression’ to determine a specific value for incrementing the counter. If you use ‘Break’, this exits the ‘For Next’ loop immediately and continues with the line following to the ‘Next’ statement. This is usually used in connection with an ‘If’ clause (If something Then Break) to exit the loop before its specified end.

The ‘Next’ statement is the lower end of the loop and increments ‘counter’ by the value given by ‘stepExpression’ or by 1 if no ‘stepExpression’ incrementation is defined. The loop repeats as long as the value of ‘counter’ is equals the value of ‘stop.’

Notes:

The speed of ‘For Next’ loops depends on the variable types used. ‘For Next’ loops run fastest when ‘counter’ is an integer variable and ‘start,’ ‘stop,’ and ‘stepExpression’ are integer expressions.

Count backwards using ‘stepExpression’ with a negative incrementation value. Take extra care when nesting ‘For Next’ loops.

Remember to end the loop last initiated (see example) or an infinite loop occurs. Example 1:

```
Dim ctr As Integer
For ctr = 1 To 5
  Log("Z")
Next
```

For Each

In the future, not implemented yet:

If you use Objective-Basic Framework classes like ‘Array’ or ‘Dictionary’, you have the option to access all elements of those containers, which the more modern style syntax ‘For Each’. See the examples of ‘Array’ for more details.

Other kind of loops

Use the following loops when you are not sure how often a command should be repeated: ‘Do’, ‘While’, ‘Loop’. There is only one way to use the keyword ‘While’ in order to test a condition within a ‘Do...Loop’-statement.

You can test the condition before the commands inside the loop are executed or you can test the condition after the commands of the loop have been executed at least once. If the condition is ‘Yes’ (in the following procedure ‘SubBefore’) the commands inside the loop execute. If you set ‘myNumber’ to 9 instead of 20, no command inside the loop execute. Inside procedure ‘SubAfter,’ all commands execute at least once because the condition is not true.

```
Sub SubBefore()
  Dim Counter As Integer = 0
  Dim myNumber As Integer = 20
  Do While myNumber > 10
    myNumber = myNumber - 1
    Counter = Counter + 1
  Loop
```

```
End Sub
```

Do While...Loop

A ‘Do While’ loop is a group of statements enabling you to define a loop that will repeat until a certain condition remains ‘Yes’.

‘Do’: launches the loop and must be the first statement ‘Loop’: ends the loop and must be the last statement ‘While’: lets the loop repeat while the condition is ‘Yes’ Condition: a numeric or string expression with a result of ‘Yes’ or ‘No’ ‘Break’: exits the loop at this very line and lets the program continue behind the ‘Loop’-statement ‘Continue’: jumps directly to the ‘Loop’-condition

Syntax:

```
Do While Expression
  [Statements]
Loop
```

Examples: In the first example, the loop repeats as long as ‘xyz’ remains 5:

```
Do While xyz = 5
  (lines of code)
Loop
```

Please note the lines of code will never be executed if ‘xyz’ is not 5 when entering the loop. You may use the condition with a number of expressions like 'And' and 'Or':

```
Do While x < 10 And y < 10
  (lines of code)
Loop
```

The loops repeats while x and y are smaller than 10.

Note: Please be careful when nesting several loops within each other.

Good programming practice would recommend using separate variables for each loop. The same might happen with a ‘For...Next’

loop within:

```
Do While counter < 10
  (lines of code)
  For counter = 1 To 5
    (lines of code)
  Next
```

Loop Moreover, be careful not to interchange the different loops:

```
Do While counter < 10
  (lines of code)
  For i = 1 TO 5
    (lines of code)
  Loop
Next i
```

This results in an error because ‘Loop’ has to appear after ‘Next.’

Explicitly leave a loop

Normally, a loop ends when its condition allows it. Sometimes it might be useful to exit a loop before the condition is met. Manually exit a loop using the ‘Break’-statement.

‘Break’ – leave a for loop

Syntax:

```
Break
```

Explicitly test a condition

A loop condition is tested after every loop iteration. However, it might be useful to test the loop condition earlier. Manually test a loop condition using the ‘Continue’-statement. As a result, the current loop condition is tested.

‘Continue’ – manually test loop condition Syntax:

```
Continue
```

Nested control structures

Embed control structures into other control structures by putting a ‘For’-loop into an ‘If’-statement. This is called nested control structures. You can embed as many control structures as needed, but you should indent every control structure line so that your code is easier to read. The examples in this manual are always intended and formatted.

Procedures / Methods

Your programs have been short, each designed to demonstrate a single programming technique. When you start writing real programs, however, you will discover they can grow to many pages of code. When programs increase in length, they become harder to organize and read. Professional programmers use modular programming to decrease the length of programs. Modular programming uses procedures. A procedure is like a small program within your main program. Objective-Basic source code is inside a procedure, normally. A procedure is a set of commands inside the written words ‘Sub’ and ‘End Sub’ or ‘Function’ and ‘End Function’. There are different types of procedures. In fact, your program contains of methods only, declared inside of classes. There is no main program at all, like in old BASIC languages.

- Sub (inside a class) - Sub-procedures contain commands and statements but do not return a value or cannot be used in expressions.
- Functions (inside a class) - Function-procedures contain commands and statements. Function-procedures always return a value, e.g. the result of a complex math operation. Because functions return values, they can be used in expressions. Functions like subs can have arguments.
- Event-procedures are always function-procedures. An event procedure is related to a control. When the runtime of Cocoa notices an event, it calls the related event-procedure, if it is

connected.

Sub-Procedure

A sub-procedure can have arguments, e.g. variables, expressions, or constants that are given to the sub-procedure while calling it.

Syntax:

```
Sub Name([Arguments])
  [Statements]
End Sub

Sub Name([Arguments])
  [Statements]
End Sub
```

Function-Procedure

A sub-procedure can have arguments, variables, expressions, or constants that are given to the sub-procedure when calling it. Function-procedures return values.

Syntax:

```
Function Name([Arguments]) [As Type]
  [Statements]
End Function
```

Arguments

For all practical purposes, a procedure can have as many arguments as needed. You must be sure the arguments you specify in the procedure's call exactly match the type and order of the arguments in the procedure's sub line. To use more than one argument in a procedure, separate the arguments with commas. You can pass one or more arguments to a procedure. Keep in mind the arguments are passed to the procedure in the order in which they appear in the procedure call. The procedure's sub line must list the arguments in the same order they are listed in the procedure call.

If a procedure does not have arguments, you must not write any expression or variable inside the braces when calling it. All statements and commands are executed only after calling the procedure. Arguments have names. If you have many arguments, you can separate them by a comma (,). Every argument is like a variable declaration and leads to automatically declared variables when the statements and commands of a procedure are executed.

Syntax of arguments:

```
Name As Type
Name [As Type]
```

Named arguments

There are two ways to give arguments to a procedure. One is to name the arguments, the other is the position relevant (default) one. Arguments – the normal way:

```
PassArg(Frank", 26, 22879)
```

Named arguments:

```
PassArg(strName := "Frank", intAge := 26, birthDate := 22879)
```

If you use named arguments, you must list the arguments in the right order. It is probably more readable.

A named argument contains the argument name, colon (:) and equal sign, and the expression the argument should have.

Writing function-procedure

A function-procedure contains commands and statements, which are after ‘Function’ and before ‘End Function.’ A function-procedure is like a sub-procedure but it can return a value. That is the only difference between them. A function-procedure can also have arguments. You must use the keyword ‘Return’ for return values.

Call of sub-procedures and function-procedures

To call a sub-procedure within another procedure, write the name of the subprocedure to call and all needed arguments.

Add comments to a procedure

When creating a new procedure or changing code, comment the new or changed code. Comments have no effect on the program when it is executed, they only help other developers understand the code.

Comments start with (’). This character tells Objective-Basic to ignore all text until reaching the end of line. Find more information about comments and other comment styles in the previous chapter about comments.

Hints for calling procedures

When changing the name of classes, be sure to change all places the class is used or Objective-Basic runs into an error. You could use the file replace in the Objective-Basic menu.

To avoid name collisions, give your procedures unique names.

Leave procedure

You can leave the procedure at any line inside of a procedure. For that, use the keywords ‘Return’.

Syntax:

```
Return
```

Use of return value after calling a function-procedure

In order to get the return value of a function-procedure, you need a variable.

Example:

```
answer3 = myMsgBox("Are you satisfied with your income?")
```

If you do not need the return value, ignore it and use the function-procedure like an ordinary sub-procedure.

Example:

```
myMsgBox("Task done!")
```

Writing recursive procedures

Procedures have limited memory, so if a procedure calls itself, it consumes memory again. When procedures call themselves they are called recursive procedures.

Example:

```
Function doError(Maximum)
  doError = doError(Maximum)
End Function
```

These errors can be hidden when two procedures are calling each other once and once again or if no break out condition is used. In some situations recursive procedures might be useful.

Example:

```
Function Fakulty (N As Double)
  If N <= 1 Then ' end of recursive
    Return 1 ' (N = 0) no more calls
  Else ' Fakulty is called again
    ' wenn N > 0.
    Return Fakulty(N - 1) * N
  End If
End Function
```

Carefully test a recursive procedure to make sure if it works as expected. If an error occurs, check the break out condition. Try to minimize memory consumption:

- Release unused variables

• Check the logic of the procedure. Use loops inside loops instead of recursion calls.

Functions

Functions are very similar to subs in that both are procedures, but functions return a value and can be used in expressions.

Syntax:

```
Function Name([Arguments])[As Type]
  [Statements]
End Function
```

The arguments of a function-procedure are used as the arguments of a sub procedure. Define a function-procedure by using the keyword 'Function.' You should also define a return type for every function and normally you does define the return type explicitly. For more information about using procedures see the previous chapter.

Function Return

If you would like to set the return value of a function and exit the function, use 'Return'. With this statement you immediately leave the function.

Syntax:

```
Return Expression
```

Modifiers / Scopes

Modifiers and scopes are a very important aspect of the Objective-Basic programming language and of modern programming languages in general. Scopes tell Objective-Basic when a variable, a constant, or a procedure can be used, from which place in the source code files, actually. There are different scopes (abstract places in code):

- Private class scope
- Public class scope

Scopes are useful when you would like to organize your program or want to avoid name collisions of variables.

When you refer to a variable within your method definitions, Objective-Basic checks for a definition of that variable first in the current scope, then in the outer scopes of the current method definition. If that variable is not a local variable, Objective-Basic then checks for a definition of that variable as an instance in the current class.

Because of the way Objective-Basic checks for the scope of a given variable, it is possible for you to create a variable in a lower scope such that a definition of that same variable hides the original value of that variable. This can introduce subtle and confusing bugs into your code. The easiest way to get around this problem is to make sure you do not use the same names for local variables as you do for instance variables. Another way to get around this particular problem, however, is to use 'Me.variableName' to refer to the instance variable and just variable to refer to the local variable. By referring explicitly to the instance variable by its object scope you avoid the conflict.

Local scope

A variable declared inside a procedure is not usable outside the procedure, only the code of the procedure in which the variable has been declared can use it. The following example demonstrates how a local variable may be used. There are two procedures, one of them has a variable declared:

```
Sub localVar()
  Dim str As String
  str = "This variable can only be used inside this procedure"
  Log(str)
End Sub

Sub outOfScope()
  Log(str) ' causes an parser error
End Sub
```

Classes

A simple application can contain a simple interface, while all source code is inside a class file. Create a class, which contains a methods, which is useful for your interface.

You Objective-Basic code is stored in classes. You can archive your code within classes. Every class consists of the declaration part and the methods you have inserted.

A class can contain:

- Declarations - for variables, types, enumerations and constants
- Methods (also called procedures) - which are not assigned to a special event or object. You can create as many procedures as you want.

You cannot put several classes in one file.

Classes unlike procedures are not executed

A class consists of a declaration part only; you must create the methods yourself. Additionally, a class contains no main program. Methods are executable inside classes and execute if the proper event is raised or if another part of your program has called a method of that class.

Edit Class

It is not much different editing text in a word processor or in Objective-Basic's IDE. You have a cursor showing the current position you can type. You can also find and replace inside your source code as in a word processor.

The Objective-Basic development environment

The Objective-Basic development environment contains windows, a toolbar, and an editor that make developing your Objective-Basic

application easy. It is actually an integrated development environment (IDE). When a development environment is said to be integrated, the tools in the environment work together.

For project management you should use Finder (Software created by Apple) already installed on your Mac. A project is related to a directory containing all files of that project. The interface (windows, controls and so on) are created using the Xcode's Interface Builder already installed on your Mac (Software created by Apple). So you have the same controls like Objective-C coders have. In fact, you can directly use Cocoa as Objective-C developers can.

For example, the compiler might find an error in the source code. In addition to displaying the error, Objective-Basic opens the source file in the text editor and jumps to the exact line in the source code where the error occurred.

A large part of application development involves adding and arranging visual elements on interfaces. Interface Builder is a great tool to make designing interfaces a simple process.

Windows

Objective-Basic windows are used to develop your programs, including monitoring the status of your projects at any time. These windows/programs include:

- Xcode's Interface Builder – uses drag and drop controls, as well as arranges them in this main development area. It is the primary tool you use to create your programs.
- Project list – works with the different parts of your project. Its views include objects and files.
- Source code editor – where you add and customize source code for your project in any phase of the development cycle.

Toolbar

Objective-Basic provides most common commands for quick access on a toolbar.

Editor

Objective-Basic has one editor for creating and managing Objective-Basic projects. You can use this editor to control the development of your projects, such as editing source code and manipulating classes.

Classes and objects of Objective-Basic

There are two types of objects in Objective-Basic: visual objects and invisible objects.

- A visual object is a control, is visible at runtime, and lets users interact with your application. It has a screen position, a size, and a foreground color. Examples of visual objects are buttons.
- An invisible object is not visible at runtime. Some objects can contain other components, such as an application window containing a button. With Interface Builder, you add visual objects/controls to your forms to assemble applications.

Projects

Projects keep your work together. When developing an application in Objective- Basic, you work mainly with projects. A project is a collection of files that make up your Objective-Basic application. You create a project to manage and organize these files. The project list shows each item in a project. Starting a new application with Objective-Basic begins with the creation of a project. So before you can construct an application with Objective-Basic, you need to create a new project. Normally, you do this by copying an existing project's directory to a new location and changes the files you need. You can add and remove files by using Finder or in Objective-Basic IDE itself.

Interfaces

In your Objective-Basic-application, interfaces are not only masks for inputting and changing data but they are the graphical interface of your application. In the eyes of the beholder, they are the application! By creating your application using interfaces, you control the program flow with events that are raised in the forms.

Appendix

Argument

A value that is passed to a procedure or function. Also see Parameter.

Arithmetic Operations

Mathematical operations such as addition, multiplication, subtraction, and division that produce numerical results.

Array

A variable that stores a series of values accessed using a subscript.

BASIC

Beginner's All-Purpose Symbolic Instruction Code, the computer language upon which Objective-Basic is based.

Bit

The smallest piece of information a computer can hold. A bit can be only one of two values, 0 or 1.

Boolean

A data type representing a value of true or false.

Boolean Expression

A combination of terms that evaluates to a value of true or false. For example, (x = 5) and (y = 6).

Branching

When program execution jumps from one point in a program to another, rather than continuing to execute instructions in strict order. Also see Conditional Branch and Unconditional Branch.

Byte

A piece of data made up of eight bits. Also see Bit.

Compiler

A programming tool that converts a program's source code into an executable file. Objective-Basic automatically employs a compiler when you run a program or select the File menu's Make command to convert the program to an executable file. Also see Interpreter.

Constant

A predefined value that never changes.

Data Type

The various types of values that a program can store. These values include Integer, Long, String, Single, Double, and Boolean and so on.

Decrement

Decreasing the value of a variable, usually by 1. Also see Increment.

Double

The data type that represents the most accurate floating-point value, also known as a double-precision floating-point value. Also see Floating Point and Single.

Empty String

A string that has a length of 0, denoted in a program by two double quotes. For example, the following example sets a variable called str1 to an empty string: str1 = "".

Event (or Action)

A message that is sent to a program as a result of some interaction between the user and the program.

Executable File

A file, usually an application, that the computer can load and run.

File

A named set of data on a disk.

Floating Point

A numerical value that has a decimal portion. For example, 12.75 and 235.7584 are floating-point values. Also see Double and Single.

Function

A subprogram that processes data in some way and returns a single value representing the result of the processing.

Increment

Increasing the value of a variable, usually by 1. Also see Decrement.

Infinite Loop

A loop that can't end because its conditional expression can never evaluate to true. An infinite loop ends only when the user terminates the program. Also see Loop and

Integer

A data type representing whole numbers between –2,147,483,648 to 2,147,483,647. The values 240, –128, and 2 are examples of integers. Also see Long.

Literal

A value in a program that is stated literally. That is, the value is not stored in a variable.

Local Variable

A variable accessed only from within the subprogram in which it is declared. Also see Variable Scope.

Logic Error

A programming error that results when a program performs a different task than the programmer thought he programmed it to perform. For example, the program line If X = 5 Then Y = 6 is a logical error if the

variable X can never equal 5. Also see Runtime Error.

Logical Operator

A symbol comparing two expressions and resulting in a Boolean value (a value of true or false). For example, in the line if X = 5 and Y = 10 then Z = 1, and is the logical operator. Also see Relational Operator.

Long

A data type that represents very large integer values. Also see Integer.

Loop

A block of source code that executes repeatedly until a certain condition is met. Loop Control Variable A variable holding the value that determines whether a loop continues to execute.

Machine Language

The only language a computer truly understands. All program source code must be converted to machine language before the computer can run the program.

Mathematical Expressions

A set of terms using arithmetic operators to produce a numerical value. For example, the terms (X + 17) / (Y + 22) make up a mathematical expression. Also see Arithmetic Operations.

Method

A procedure associated with an object or control that represents an ability of the object or control. For example, a Command Button's Move method repositions the button on the form.

Numerical Literal

A literal value that represents a number, such as 125 or 34.87. Also see Literal and String Literal.

Numerical Value

A value that represents a number. This value can be a literal, a variable, or the result of an arithmetic operation.

Object

Generally, any piece of data in a program. Specifically in Objective-Basic, a set of properties and methods that represent some sort of real-world object or abstract idea.

Order of Operations

The order in which Objective-Basic resolves arithmetic operations. For example, in the mathematical expression (X + 5) / (Y + 2), Objective-Basic performs the two additions before the division. If the parentheses had been left off, such as in the expression X + 5 / Y + 2, Objective-Basic would first divide 5 by Y, then perform the remaining addition operations.

Parameter

Often meaning the same as “argument,” although some people differentiate argument and parameter where an argument is the value sent to a procedure or function and a parameter is the variable in the function or procedure that receives the argument. Also see Argument.

Procedure

A subprogram performing a task in a program but does not return a value. Also see Function.

Program

A list of instructions for a computer.

Programming Language

A set of English-like keywords and symbols that enable a programmer to write a program without using machine language.

Program Flow

The order in which a computer executes program statements.

Relational Operator

A symbol that determines the relationship between two expressions. For example, in the expression X > 10, the relational operator is >, which means “greater than.” Also see Logical Operator.

Return Value

The value a function sends back to the statement that called the function. Also see Function.

Runtime Error

An system error that occurs while a program is running. An example is a divide-byzero error or a type-mismatch error. Without error handling such as that provided by the Try/Catch statement, runtime errors often result in a program crash.

Scope

See Variable Scope.

Single

The data type that represents the least accurate floating-point value, also known as a single-precision floating-point value. Also see Double and Floating Point.

Source Code

The lines of commands making up a program.

String

A data type that represents one or more text characters. For example, in the assignment statement str1 = "I’m a string," the variable str1 must be of the String data type.

String Literal

One or more text characters enclosed in double quotes.

Subprogram

A block of source code that performs a specific part of a larger task. In Objective- Basic, a subprogram can be either a procedure or a function. Also see Function and Procedure.

Unconditional Branch

When program execution branches to another location regardless of any conditions. An example of an unconditional branch is the GoTo statement.

User Interface

The visible representation of an application, usually made up of various types of controls, that enables the user to interact with the application.

Variable

A named value in a program. This value can be assigned and reassigned a value of the appropriate data type.

Variable Scope

The area of a program in which a variable can be accessed. For example, the scope of a global variable is anywhere within the program, whereas the scope of a local variable is limited to the procedure or function that declares the variable. Also see Local Variable.

Application Bundles

(APP directories)

On Mac OS X, a bundle is a directory that allows related resources such as software code to be grouped together.

Application bundles are often presented to users as a single file known as a “package”. This file is really a directory ending in a .app extension. Control-clicking (or right-clicking) on the package allows a user to open up the bundle and see the contents. In an application, the first directory in the bundle is usually Contents; within contents there is usually another directory with the executable code (called

MacOS for Macs), which contains the application's executable code, and a directory called Resources, which contains the resources of the application.

PackageMaker

What is PackageMaker? PackageMaker is the installing solution provided by Apple with its developer tools. PackageMaker is installed with the Developer Tools. So if you don't have installed the Developer Tools, you won't find it.

Copyright

Copyright © 2007 - 2012 by www.objective-basic.com.

Products named on this website are trademarks of their respective owners.