Delphi Quick Reference Card 1.01

http://www.explainth.at

Color key overleaf

Unit Structure

unit unitName;
interface
[uses UnitA,UnitB...;
class declarations[1]
exports routineA,routineB...;[2]
var Variables][3]
implementation
[{$R *.dfm}][1]
{$R WindowsXP.res}[4]
{$R resourceName.res}[5]
exports routineA,routineB...;[2]
uses UnitAA,UnitBB...;
var Variables;[5]**]**
Code...
[initialization Initialization Code;]
[initialization Initialization Code;
finalization Finalization Code;[6]**]**
end.

1. In a form this includes one form and the $R *.dfm statement below is obligatory.
2. To export routines from a library with unit in its uses clause.
3. Visible in all units that use the present unit
4. For WindowsXP style UI effects.
5. Any custom resources used.
6. Visible within the unit
7. Initialization/finalization code can be a procedure call. No finalization without initialization but just a blank initialization statement is enough.

Names & Notation

As a general rule all identifier names – i.e. names for units, controls, objects, variables... - must be alphanumeric or the _ character. The first character cannot be a number.

There is no single accepted notation standard. We suggest the following

- Hungarian style notation for control/component & interface identifiers. e.g. **btnName** for a TButton control with **Name** describing its function.
- **i,j,k**... for generic integer variables used as loop counters etc.
- Javascript style descriptive camel capitalized names for all other variables. e.g. intRate.

Names are not case sensitive.

Visibility, Scope & Garbage Collection

Variables declared inside a routine are only visible within the routine – and to nested routines.

Declarations using the **var** keyword in the **interface** section of a unit are visible within the unit and wherever the unit is present in a **uses** clause.

Declarations using the **var** keyword in the **implementation** section of a unit are visible within the unit.

Objects implementing interfaces are reference counted. They are destroyed when their reference count reaches zero. All other objects and any allocated memory must be explicitly destroyed/released after use.

Variable Types[1]

Type	Size	Range
Boolean	1	false or true
Byte	1	0..255
Cardinal	4	0..4294967295
Char	1	Extended ASCII
Currency	8	±9.22E14
Double	8	5E-324..1.7E308
Extended	10	3.6E-4951..1.1E4932
Integer	4	-2147483648..2147483647
Int64	8	$-2^{63}..2^{63} - 1$
PChar	4+[2]	pointer to array of char
Pointer	4	Generic Pointer
P#	4	# is Integer, Double, etc
PWideChar	4+[3]	pointer to arry of widechar
Set	32	See below[4]
String	4+[2]	string of char
TDateTime	8	See below[5]
WideChar	2	Unicode Character
WideString	4+[6]	string of Unicode characters
Word	2	0..65535

1. Other types exist.
2. 4 + length of string + 1
3. 4 + 2 bytes per stored widechar
4. To store bytes, chars, enumerations with < 256 members etc.
5. 0 is 12:00 am, 12/30/1899. No values between -1 & 0. Fraction represents time of day, e.g. 0.25 = 6:00 am. For dates prior to 12/30/1899, add time of day to absolute value of day, e.g. -1.25 for 6:00 am 12/29/1899.
6. 4+ twice length of string + 1

Special Constants

false, **true**, **nil**, **MAXWORD**, **MAXINT**, **MAXDOUBLE**, **MINDOUBLE** etc.

Enumerations

e.g. type TDelphiVersion = (dv5[= 5],dv6,dv7,dv8)

Enumerations can be manipulated using **inc**, **dec**, **pred** and **succ**. **ord** can be used to get their ordinal value. Prepend enumeration members with two or more lowercase letters identifying their parent enumeration.

Enumerated values require one or more depending on the number of members in the parent enumeration.

Array Types

Any ordinal type can be used to define an array type. e.g.

- **TVersions** = **array**[**TDelphiVersions**] **of String;**
- **TLevels** = **array**[**-3..3**] **of Integer;**
- **TLetters** = **array**[**'a'..'z'**] **of Char;**
- **TInfo = array[Boolean,0..9] of PChar;**

Operators

Operator	Example	Result
+	3 + 2 'explain' + 'that'	5 explainthat
-	3 - 2	-1
*	3*2	6
/	3/2	1.5
div	3 div 2	1
mod	3 div 2	3 – (3 div 2)*2
:=	i:=2	Assignment
=	3 = 3 2 = 3	true false
<	3 < 4	true
<=	2 <= 3	true
>	'explain' > 'Explain'	true
>=	5 >= 4	true
<>	2 <> 3 'explain' <> 'explain'	true false
shl	1 shl 2	4
shr	2 shr 1	1
i:=2;j:=7		
and	(i < 3) AND (j >= 5) i AND j	true 2
or	(i < 3) or (j > 7) i OR j	false 7
xor	(i < 3) XOR (j = 5) (i < 3) XOR (j = 7) (i > 2) XOR (j < 7) i XOR j	true false false 5
not	(i < 3) AND NOT(j > 7) not(i)	true -3
s1:=[1..3];s2:=[3..7]		
+	s1 + s2	[1..7]
-	s1 – s2 s2 - s1	[1,2] [4..7]
*	s1*s2	[3]
s1:=[1..3];s2:=[1,2,3];s3:=[1..7]		
<=	s1 <= s3	true
>=	s3 >= s2	true
=	s1 = s2	true
<>	s1 <> s2	false
in	4 in s1 5 in s3	false true
exclude	exclude(s1,3)	[1,2]
include	include(s1,9)	[1,2,9]

Conversion from Strings[ii]

StrToCurDef(s,def) – s to currency. **def** on error.
StrToInt[64]Def(s,def) – **s** to integer. **def** on error.
StrToFloatDef(s,def) – **s** to real. **def** on error.
StrToDateTimeDef(s,def) – **s** to datetime. **def** on error.
val(S,V,Code) – **s** converted to number & stored in **V**. **Code** > 0 indicates position in **s** of first error.

Conversions to Strings[ii]

FloatToStr(value) – **value** as a string.
Format(ptrn,[arg1,arg2...])* - uses **ptrn** to build a string. **%d**, **%f** etc in pattern are replaced by values in **args**.
FormatDateTime(ptrn,datetime) – returns **datetime** as string formatted using **ptrn**. If **ptrn** is empty uses short date format.
FormatFloat(ptrn,value) – returns **value** as string formatted using **ptrn**.
IntToHex(value,N) – **value** in hexadecimal with **N** digits
IntToStr(value) – **value** as a string.

Date & Time Routines[ii]

Date – current date, time fraction set to zero.
DateTimeToStr(d) – d to string using locale.
DecodeDate(Date,Y,M,D) – year, month & day to YMD
DecodeTime(Date,H,M,S,N) – hrs, mins, s & ms to HMSN
EncodeDate(Y,M,D) – returns datetime value.
EncodeTime(H,M,S,N) – returns time fraction of datetime.
FormatDateTime(Format,Date) – returns formatted date string

Delphi Quick Reference Card1.01

Drive/File/Folder Manipulation[ii]

ChangeFileExt(AFile,AExt) – returns filename with new extension. **AExt** must include the **.** character.
***System.*ChDir(dir)** – changes current directory.
CreateDir(dir) – creates directory. **false** on error.
***SysUtils.*DirectoryExists(dir)** – **true** if **dir** exists.
***SysUtils.*DiskFree(drive)** – free bytes on drive. 0 = current, 1 = **A** etc.
ExtractFileExt(AFile) – returns .ext.
ExtractFileName(AFile) – returns filename.ext.
ExtractFilePath(AFile) – returns everything before filename.ext.
ForceDirectories(path) – creates all directories in **path**. **false** on error.
***System.*GetDir** – current directory.
RemoveDir(dir) – removes **dir**.

Execution/Flow Control

***SysUitls.*abort –** raise silent exception
break - break from loop (**for**, **repeat** or **while**)
continue – continue to next iteration of loop
exit – exit from current procedure
halt – immediate termination of program

Number Manipulation[iii]

abs - returns absolute value
***Math.*ceil(arg)** – lowest integer >= arg
exp(N) – returns e^N
***Math.*floor(arg)** – highest integer <= arg
frac(N) – fractional part of N
int(N) - integer part of real number N
***Math.*log10(N)** – log to the base 10 of N
***Math.*log2(N)** – log to the base 2 of N
Random – random number in the range 0..1
Randomize – initialize random number generator
RandSeed – Seed value for random number generator.
Round(N) – round **N** to nearest whole number. Midway values rounded to even number.
***Math.*RoundTo(N,d)** – round **N** to 10^d

Ordinal Manipulation

dec(arg,N) – decrements ordinal **arg** by **N**
high(arg) – high bound of **arg** type.
inc(arg,N) – increments ordinal **arg** by **N**
low(arg) – low bound of **arg** type.
ord(arg) – ordinal value of boolean, char or enumerated **arg**.
pred(arg) – predecessor of ordinal type **arg.**
succ(arg) – subsequent value of ordinal type **arg.**

String Manipulation[iii]

chr(arg) – ASCII character at **arg.**
SysUtils.*CompareStr(s1,s2) – case sensitive comparison. s1 < s2 returns -1;s1 = s2 returns 0 & s1 > s2 returns 1.
SysUtils.*CompareText(s1,s2) – case insensitive comparision. Returns as above.
Copy(s,Index,Count) – Count characters in **s** starting from **Index**.
Delete(s,Index,Count) – deletes **Count** characters in **s** starting at **Index**.
***StrUtils.*LeftStr(s,Count)** – **Count** characters in **s** starting from the left. **RightStr** is similar.
***StrUtils.*MidStr(s,Index,Count)** – **Count** characters in **s** starting from **Index**.
Length(s) – number of characters in **s**.
SysUtils.*LowerCase(s) – **s** in lower case. **UpperCase** is similar.
SysUtils.*SameText(s1,s2) – returns true if s1 = s2, not case sensitive. Returns **true** or **false**.
SetLength(s,len) – sets length of string **s** to **len.**
StringOfChar(Char,Count) - returns string containing **Count Char**s.
UpCase(c) – character **c** in uppercase.

Variant Manipulation[iv]

VarFromDateTime(date) – **date** as a variant.
VarToDateTime(V) – **V** as TDateTime.
VarAsType(V,AType) – **V** converted to variant of type **AType**.
VarToStr(V) – **V** as a string.
VarToWideStr(V) – **V** as a widestring.
VarType(V) – variant type of **V**.

Format Specifiers

DateTime Formats

- **c** – ShortDateFormat
- **d** – day, no leading zero.
- **dd** – day, leading zero if necessary
- **ddd** – Short day names
- **dddd** – Long day names
- **m**, **mm**, **mmm**, **mmmm** – Month names, as above.
- **yy** - two digit year
- **yyyy** – four digit year.
- **h**, **n**, **s** – hour, minute & second. No leading zero.
- **hh**, **nn**,**ss**- hour, minute & second with leading zero
- **t** – ShortTimeFormat
- **tt** – LongTimeFormat
- **am/pm** – Use 12h clock. Follow **h**|**hh** by **am** or **pm**
- **ampm –** use 12h clock. Follow **h**|**hh** by TimeAM| PMString global variables.
- / date separator as in DateSeparator global variable
- : time separator as in TimeSeparator global variable.
- **'xx' or "xx" -** literal characters

Format function specifiers

Format strings consist of one or more specifiers bearing the form **%[-][w].[d]L** where

- **-** indicates left justification. (The default is right)
- **w** indicates the total character width of the output value. If necessary this is padded out with spaces – right or left depending on the justification specifier.
- **d** is the precision specifier. The meaning of this depends on the nature of the quantity being formatted.
  - The number of characters in integers & hexadecimal integers.
  - The number of decimals in real numbers in general , f, format.
  - The number of decimals + the **E** in real numbers in scientific format.
  - The number of characters in a string.
- **L** indicates that nature. **d** for integer, **f** for real, **e** for scientific, **n** real but with thousands separators, **s** for string and **x** for hexadecimal integer.

Example	Function
%d	Simple Integer formatting
%0.nd	Integer with **n** digits – padded if shorter
%m.nd	Integer with **n** digits in a width of **m**. m is ignored if insufficient.
%m.nf	Floating point number, width **m** with **n** decimal digits.
%-m.nf	As above but left justified.
%m.ns	String formatted to a width of **m** characters and containing **n** characters. Truncated if **n** is less than string length. **n** is ignored if greater than string length.
%m.nx	Integer in hexadecimal format. Rest as for %d, above.

Other options exist.

Conditional Execution/Brancing

Multiline if..then..else

```
if Condition then
begin
   Code
end[ else
begin
   Code
end];
```

Single line if..then..else

```
if Condition then Code else Code;
```

```
case selector of
  caseList1:code;
  caseList2:code;
   …
  caseListn:code;
  [else code;]
end;
```

selector can be any ordinal type. ***code*** can be a function/procedure call.

Looping

```
for i:=LowBound to HighBound do
begin
   Code;
end;

for i:=HighBound downto LowBound do
begin
   Code;
end;

repeat
   Code;
until  Condition;

while Codition
begin
   Code;
end;
```

Dispense with the **begin** & **end** to execute a single line of code. repeat loops execute at least once. Use **break**, **continue** or **exit** to modify/terminate loop execution.

Notes

[i] **–** MAXDOUBLE etc are defined in ***Math***.
[ii] **–** Unless preceded by ***Unit.***, the routine is in **SysUtils**
[iii] **–** Unless preceded by ***Unit.***, the routine is in **System**
[iv] **–** Unless preceded by ***Unit.***, the routine is in **Variants**
* For widestrings use the same function but preceeded by Wide, e.g. **WideFormat**.

Color Codes

blue – Delphi keyword
green – Delphi routine (function or procedure)
[option] - optional
Math. - unit to be specified in **uses** clause. Does not apply to ***System***.

An extensive range of free quick reference cards is available at http://www.explainth.at