

## ZIM 9.10

## **Programming Basics**



### What is Zim?

#### Zim is

a complete framework to develop and run professional and mission critical applications by tightly integrating a lean relational database, a powerful Fourth Generation Language, an integrated development tool, the integration with outside world and client user interfaces.



#### **Building Zim Applications**

A full Zim application is a set of Zim Documents (text files) containing Zim commands organized as procedure programs (structured in formal procedures) or macro programs (no formal procedures).

Any Zim program can be created and edited using ZimIDE.

As soon as the program is created, it can be run (with or without a compilation) by ZimQTC or invoked by another program.

All Zim commands contained in a program can be run individually in ZimQTC's prompt with the obvious exceptions of the ones requiring flow control.



#### **Executing Macro Programs**

```
DocName Parameter-1 Parameter-2 ...
```

**Docname** is the name of the document that contains the macro program;

**\*Parameter-n\*** is one or more expressions used to initialize the local macros #**<1>**, #**<2>**, etc., until #**<9>**.

```
FixError
RunExample 3500 vVarCode "Description of Variables"
```

```
This is a Macro Program example (RunExample)
Find all EMPLOYEES Where MonthlySalary > #<1> -> sEmp1
List all sEmp1 format "#<3>"
sort sEmp1 by "#<2>"
List all sEmp1
```



## **Procedures Programs**

Procedure programs use the formal mechanism of the Zim PROCEDURE command to pass values to the program and/or receive values from the program.

It may contain zero or more local procedures followed by the main procedure itself.

```
[LOCALPROCEDURE LocName ([parameters]) [LOCAL (local_vars)]
    [exception handlers]
    local procedure body
ENDPROCEDURE]

PROCEDURE ProcName ([parameters]) [LOCAL (local_vars)]
    [exception handlers]
    procedure body
ENDPROCEDURE
```



## **Procedures Programs**

Parameters can be of the IN, INOUT or OUT and have no explicit type but an implicit VARCHAR definition.

The main procedure and local procedures my use local variables with the same VARCHAR definition.

The main procedure name must match the name of an existing Zim Document.

The number of parameters and local variables is controlled by a configuration parameter but is set by default to 256.

The Zim command ENDPROCEDURE executes an implicit RETURN command.



## **Stopping and Pausing Programs**

**BYE** Exits an application session and returns control to the operating

system.

**STOP** Ends execution of an application program and returns to the

main prompt level or, if already at the prompt, it is equivalent to

a **BYE** command.

**PAUSE** Causes execution to be halted and a message to be output.

**HALT** In application programs, sets break points where execution is to

be halted.

**SLEEP** < seconds > Suspends the execution of an application for a specified period

of time.





## **Controlling Execution**

The flow of control of a Zim program goes from top to bottom unless the following control structures and statements intervene to change this flow.

```
% Control Structures
CASE WHEN OTHERWISE ENDCASE
IF... ELSEIF... ELSE... ENDIF
ON ENDON
WHILE ENDWHILE
% Commands
BREAK
CONTINUE
GOTO [Label] [PREVIOUS] [NEXT]
RETURN
BYE
STOP
SET EXCEPTION
```



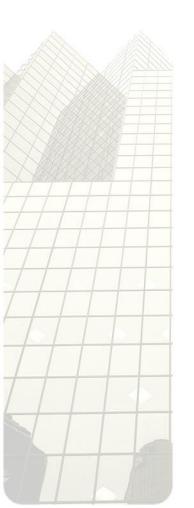
## **Controlling Execution - CASE**



```
CASE
    WHEN Event.EventName = "LostFocus"
        return
WHEN Event.EventName = "F1"
        AddNewRecord ()
WHEN salary < 20000
        ... commands ...
OTHERWISE
        GiveHelp ()
ENDCASE</pre>
```



## **Controlling Execution - IF**



```
IF Age < 18
... commands ...

ELSEIF Age between 18 and 55
... commands ...

ELSEIF Age between 56 and 65
... commands ...

ELSE
... commands ...

ELSE
ENDIF
```



## **Controlling Execution - ON**

```
procedure MyProc(...)
on break
... commands to handle the "break" condition ...
endon
... procedure commands ...
SET EXCEPTION BREAK
... procedure commands ...
endprocedure
```

The ON block will be invoked if the user presses the BREAK button or the SET EXCEPTION is executed.



## **Controlling Execution - ON**

Condition	Cause	Symptom
DEADLOCK	A transaction is aborted as result of a deadlock	\$ErrCode = 2010
BREAK	A break condition occurs at the user station	The user presses the "break" key on the keyboard
ERROR	An error or system error occurs	\$ErrLevel = 3 or 4
WARNING	A warning message occurs	\$ErrLevel = 2



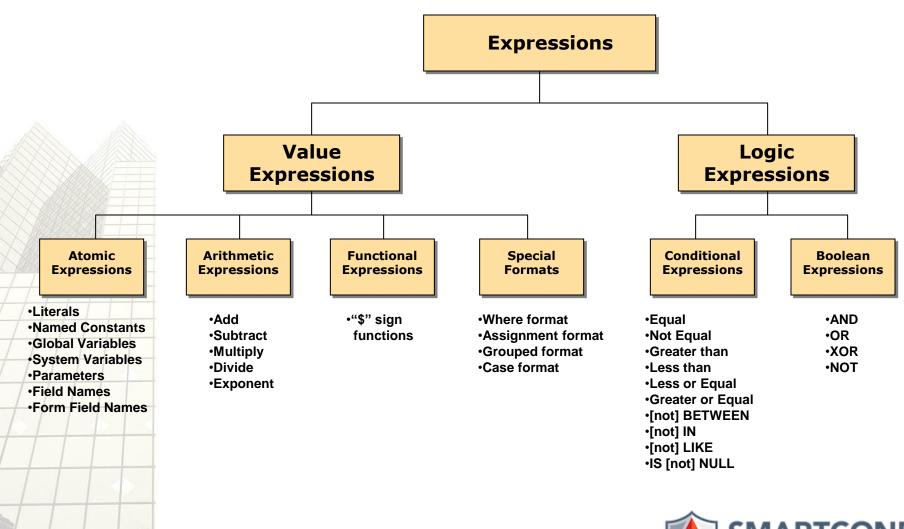
## **Controlling Execution - WHILE**



```
WHILE [logical expression]
... commands ...
if ...
break
endif
if ...
continue
endif
... commands ...
ENDWHILE
```

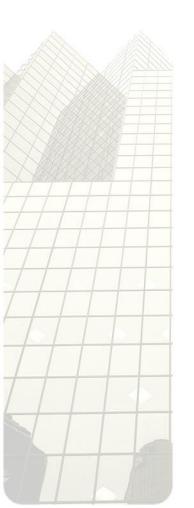


#### **Expressions**





### **Atomic Expressions**



```
123.45
'This is an Atomic Expression'
$Date
vVar1
Salary
"1500.17"
```



#### **Arithmetic Expressions**



```
Salary * 1.10
Salary + (Salary * 0.10)
(1 + InterestRate) ^ Years
```

```
5 / 2 Evaluates to 3
5 / 2.0 Evaluates to 2.5
1 + 1.01 Evaluates to 2.01
1.01 + (5 / 2) Evaluates to 3.51
$Date + 3 A date 3 days from today
```



#### **Functional Expressions**

#### Zim has hundreds of built-in functions

```
$length('Smith') -> Evaluates to 5
$cos(0) -> Evaluates to 1
$log10(2 * 50) -> Evaluates to 2
$year(20250923) -> Evaluates to 2025
$maxof($absolute(- 10), 4 + 5) -> Evaluates to 10
$toupper($substring('abcdefg', 2, 3)) -> Evaluates to 'BCD'
```



#### **Special Format Expressions - WHERE**

```
% Evaluates to $NULL ( 'A' is not = 'B')
('It is true' where 'A' = 'B')
% The expression is only evaluated if Salary > 20000
((Salary - 20000) where Salary > 20000)
List all Employees format \
   ('I work in Marketing' where DeptCode='MKT') Name DeptCode
Compute Employees where DeptCode = 'SAL' \
     evaluate \
         ((let TotSalF = $average(Salary where Gndr = 'F')), \
         (let TotSalM = $average(Salary where Gndr = 'M')), \
         (let TotSalA = $average(Salary)))
```



#### **Special Format Expressions - GROUPED**

```
((<expression1>), ..., (<last expression>))
```

The expressions are evaluated from left to right, one by one, and the value of the grouped expression becomes the last evaluated expression.

```
% Var1 evaluates to 155
% Var2 evaluates to 2
% Var3 evaluates to 20
Let Var1 = ((let Var2 = 2), (let var3 = var2 * 10), 155)
% Var1 evaluates to the Salary total but
% only records from Sales are listed.
List all Employees \
where ((let var1 = $total(Salary)), DeptCode) = 'SAL'
```



### **Special Format Expressions - CASE**

```
{<expression1> [,<expression2>] ...}
```

The expressions are any value expressions, evaluated from left to right. The result of the CASE expression will be the first expression evaluated as not null.

```
Output {"TALL" where Height > 6, 'SHORT'}
Let Salary = {Employees.Salary, 0}
let LastDay = {29 where Month = 2, \
    31 where Month in (1,3,5,7,8,10,12), 30}
```



#### **Logical Expressions**

Logical Expressions use conditional and boolean expressions to compare operands, usually value expressions, yielding a \$TRUE (i. e., 1) or \$FALSE (i. e., 0) result.

```
vA = vB OR vC NOT BETWEEN 10 AND 20
Salary > 50000 and (Let vCount = vCount + 1) < 0</pre>
```

They are evaluated from left to right, unless parenthesis explicitly determine the order of evaluation.

The Evaluation ceases as soon as the final result can be correctly determined, even if there is more expressions to evaluate.



## **Conditional Operators**

OPERATOR	CONDITION BEING EVALUATED
Expr1 = expr2	the values are equal
Expr1 <> expr2	the values are not equal
Expr1 < expr2	expr1 is less than expr2
Expr1 <= expr2	expr1 is less than or equal to expr2
Expr1 > expr2	expr1 is greater than expr2
Expr1 >= expr2	expr1 is greater than or equal to expr2
Expr1 Between Expr2 and Expr3	the value of <i>Expr1</i> is greater than or equal to <i>Expr2</i> and less than or equal to <i>Expr3</i>
Expr1 Not Between Expr2 and Expr3	the value of Expr1 is less than Expr2 or greater than Expr3
Expr1 IN (expr2, expr3,)	the value of <i>Expr1</i> is a member of the list of values ( <i>expr2</i> , <i>expr3</i> ,)
Expr1 Not IN (expr2, expr3,)	the value of <i>Expr1</i> is NOT a member of the list of values ( <i>expr2</i> , <i>expr3</i> ,)
Expr1 IS [\$]NULL	The value of Expr1 is NULL
Expr1 IS NOT [\$]NULL	The value of Expr1 is not NULL
Expr1 LIKE pattern	The value of Expr1 matches the pattern specified
Expr1 NOT LIKE pattern	The value of Expr1 does not match the pattern specified



#### **Conditional Expressions**



Conditional Expressions use conditional operators to compare value expressions yielding a logical result \$TRUE (1) or \$FALSE (0).

```
EmpNum > 1254
FirstName = 'Smith'
EmpNum Between 10 and 25
Name LIKE '%RK%'
CityCode NOT IN ('OTT','TOR','NYC')
Name = "M"?
```



#### **Boolean Expressions**

Simple Conditional Expressions can be combined into more Complex Boolean expressions by using Boolean operators.

<b>Operator</b>	Meaning
NOT expr	Negates the logical expression to the right. If the logical expression is true, the result of the boolean expression is false and vice-versa.
Expr1 AND Expr2	ANDs two logical expressions: if both expressions are true, then the result of the Boolean expression is also true; otherwise, is false.
Expr1 OR Expr2	ORs two logical expressions: the result of the Boolean expression is always true unless all expressions are false.
Expr1 XOR Expr2	XORs two logical expressions: if either one is true, then the result of the Boolean expression is also true; if both expressions are either true or false, then the result is false.

```
DeptCode = 'SAL' AND SALARY > 50000
Name = 'M'? OR CitiCode in ('TOR','NYC')
NOT CityCode in ('NYC','LON')
```

#### **Boolean Expressions**

Operator	Rule of Precedence
NOT	NOT is evaluated first
AND	AND is evaluated next
OR, XOR	Evaluated last

Operators of equal precedence are evaluated from left to right unless parenthesis are used to explicitly determine the order of evaluation.



#### **Expressions and \$NULL**

If an arithmetic or a functional expression contains a \$NULL value expression, then whole expression is evaluated to \$NULL.

If a logic expression contains a \$NULL value expression, then it is considered to be logically FALSE.



#### **OUTPUT Command**

The OUTPUT (or OUT) evaluates expressions and outputs their results at the Zim prompt.



#### **SET Command**

There are many SET commands to change certain characteristics and behavior of the Zim application. Some of them are:

SET	OUTPUT DocName
SET	OUTPUT TERMINAL
SET	NULLVALUE <string></string>
SET	SAVE
SET	RESTORE
SET	LEXTRACE ON/OFF
SET	SINGLESTEP ON/OFF
SET	MEMBERCOUNT ON/OFF
SET	STRATEGY ON/OFF
SET	ERRORS ON/OFF

SET RUNTIME ON/OFF

- % Outputs to a document.
- % Outputs to terminal.
- % NULL values become <string>.
- % Saves the current settings.
- % Restores a previous SAVE.
- % Displays Zim program tracing.
- % Traces a program line by line.
- % Displays member count.
- % Shows access strategy.
- % Displays errors at the prompt.
- % Runs compiled programs.



#### **SYSTEM Command**

Execute OS commands within a Zim program.

```
% Copies a file on Windows.
SYSTEM "!copy c:\\Mydir\\MyFile.txt d:\\AnotherDir"
% List the contents of the current database path.
SYSTEM $concat("ls ", $dbpath, " > xxx.txt")
```





# ZIM 9.10

## **Programming Basics**

