What is VectorScript ?

• VectorScript is the scripting language component of the VectorWorks software package.

• It is a lightweight programming language which syntactically resembles **Pascal**.

• **VectorScript** is actually a “superset” of the Pascal language, extending basic Pascal capabilities with a number of APIs (application programming interfaces) which **provide access to** the features and funtionality of **the VectorWorks CAD engine.**

Some Background On VectorScript

• VectorScript originated in 1988 as **MiniPascal** in the Mini-CAD+ 1.0 release.

• With the advent of VectorWorks in 1998, MiniPascal became VectorScript.

• The core VectorScript language continues to be developed by **Nemetschek** North America.

Some Background On VectorScript

What VectorScript Can Do

• VectorScript is a relatively general purpose programming language, it provides the ability to perform **most common programming tasks**. Tasks such as
  • computations
  • storing a value, and
  • manipulating data

• VectorScript also provides **extended capabilities** specific to the **VectorWorks product**.

• **Object Creation and Editing**
  • create and edit objects directly
  • primitive objects (lines, rectangles, ...)
  • more complex objects (multiple 3D extrudes, 3D solids

• **Document Control**

• **Extended Data**
  • access to and control over - worksheets
    - data records
    - textures

Some Background On VectorScript

What VectorScript Can't Do

• VectorScript does not have the ability to work **across multiple documents** or **outside of a VectorWorks** document context.

• For reasons of simplicity and stability, VectorScript does not have the ability **to manage or control memory allocation**.

• VectorScript does not support **system level calls** for file-related or other tasks.

• VectorScript does not provide **external database** or other connectivity options.

How does VectorScripts look?

```
PROCEDURE FirstExample;
CONST
        kGREETING = 'Hello ';
VAR
        myMessage : STRING;


BEGIN
        myMessage:='VectorScript';

        Message(kGREETING,myMessage);
        Wait(5);
        SysBeep;
        ClrMessage;
END;
Run(FirstExample);
```

How does VectorScripts look?

.. in the VectorWorks VectorScript Editor.

```
PROCEDURE FirstExample;
CONST
        kGREETING = ‘Hello ‘;
VAR
        myMessage : STRING;


BEGIN
        myMessage:=’VectorScript’;

        Message(kGREETING,myMessage);
        Wait(5);
        SysBeep;
        ClrMessage;
END;
Run(FirstExample);
```

How does VectorScripts look?

.. in a texteditor with Syntax Highlighting

```
PROCEDURE FirstExample;
CONST
        kGREETING = 'Hello ';
VAR
        myMessage : STRING;


BEGIN
        myMessage:='VectorScript';

        Message(kGREETING,myMessage);
        Wait(5);
        SysBeep;
        ClrMessage;
END;
Run(FirstExample);
```

How does VectorScripts look?

the different parts of a VectorScript

```
PROCEDURE FirstExample;
CONST
        kGREETING = ‘Hello ‘;
VAR
        myMessage : STRING;


BEGIN
        myMessage:=’VectorScript’;

        Message(kGREETING,myMessage);
        Wait(5);
        SysBeep;
        ClrMessage;
END;
Run(FirstExample);
```

Identifies the script to the VectorScript compiler

How does VectorScripts look?

the different parts of a VectorScript

```
PROCEDURE FirstExample;

CONST
        kGREETING = ‘Hello ‘;
VAR
        myMessage : STRING;

BEGIN
        myMessage:=’VectorScript’;

        Message(kGREETING,myMessage);
        Wait(5);
        SysBeep;
        ClrMessage;
END;
Run(FirstExample);
```

Declares data storage for the script

The source code of the script

How does VectorScripts look?

the different parts of a VectorScript

```
PROCEDURE FirstExample;

CONST
        kGREETING = ‘Hello ‘;
VAR
        myMessage : STRING;


BEGIN
        myMessage:=’VectorScript’;

        Message(kGREETING,myMessage);
        Wait(5);
        SysBeep;
        ClrMessage;
END;
Run(FirstExample);
```

Tells the VectorScript compiler to run the script

Is VectorScript easy ?

yes!

..always ?

no.

Can I get sick of VectorScript ?

no.

..really ?

ok, .. it depends.

How to learn VectorScript?

- **write programms** with VectorScript
- **make mistakes**
- **make mistakes**
- **make mistakes**
- **make mistakes**

The Grammatik of VectorScript

Case Sensitivity

• VectorScript **is not case sensitive**. This means that items such as language keywords, variables, function names, and any other identifiers can be specified using uppercase, lowercase, or a mixed case and still be compatible with other variations of the same item.

- APFEL = apfel = Apfel

The Grammatik of VectorScript

speaking Variables

- use **speaking Variables** in your Scripts
  - it makes live more easy
  - it makes your Script more **readable**
  - you and your Colleagues will understand your script faster

  - e.g. rectLength, rectLeftCornerXpos

The Grammatik of VectorScript

space

• Since spaces, tabs, and new lines do not have meaning to the VectorScript compiler, you are free to use them to indent and format your script code. This type of formatting makes your scripts **easy to read** and **understand**.

```
PROCEDURE FirstExample;
CONST
        kGREETING = ‘Hello’;
VAR
        myMessage : STRING;

BEGIN
        myMessage:=’VectorScript’;

        Message(kGREETING,myMessage);
        Wait(5);
        SysBeep;
        ClrMessage;
END;
Run(FirstExample);
```

The Grammatik of VectorScript

space

```
PROCEDURE FirstExample;
CONST
kGREETING = ‘Hello’;
VAR
myMessage : STRING;
BEGIN
myMessage:=’VectorScript’;
Message(kGREETING,myMessage);
Wait(5);
SysBeep;
ClrMessage;
END;
Run(FirstExample);
```

The Grammatik of VectorScript

space

```
PROCEDURE FirstExample;CONST kGREETING = ‘Hello’;VAR myMessage : STRING;BEGIN myMessage:=’VectorScript’;
Message(kGREETING,myMessage);Wait(5);SysBeep;ClrMessage;END;Run(FirstExample);
```

The Grammatik of VectorScript

Comments

• Comments in VectorScript are used **to place descriptive text** within script code. They are most often used to **document script code** for your reference and for others who may work on your scripts. The Vector Script compiler ignores comments.

• The general syntax for a single VectorScript comment is:

```
{This is a comment}
```

• To comment out a block of the VectorScript code the syntax is:

```
(* my comment:
write what ever you wanted,
even VectorScriptcode or {other comments!}
 *)
```

Identifiers

• Identifiers in VectorScript are symbols which are used to refer to something else: **constants**, **variables**, **data types**, **procedure** or **function names**, and other similar items. The rules for writing VectorScript identifiers are :

- The first character must be a letter or an underscore.
- Subsequent characters may be a character, digit, or underscore.
- Identifiers may not contain spaces, tabs, or other characters.

Value Identifiers

num	color_32bit	totalLumberUsed
SUM	_dummy	A_very_fine_identifier

Invalid Identifiers

52pickup	three+two	SUB TOTAL

Reserved Words

ALLOCATE	AND	ARRAY	BEGIN
BOOLEAN	CASE	CHAR	CONST
DIV	DO	DOWNTO	DYNARRAY
ELSE	END	FALSE	FOR
FUNCTION	GOTO	HANDLE	IF
INTEGER	LABEL	LONGINT	MOD
NIL	NOT	OF	OR
OTHERWISE	PI	PROCEDURE	REAL
REPEAT	STRING	STRUCTURE	THEN
TO	TRUE	TYPE	UNTIL
USES	VAR	VECTOR	WHILE

FILE	*FORWARD*	*IMPLEMENTATION*	*INHERITED*
INTERFACE	*INTRINSIC*	*OBJECT*	*OVERRIDE*
PACKED	*PROGRAM*	*SET*	*UNIT*
USES	*WITH*		

Variables, and Constants

Variables

• The **VAR block** in the VectorScript is the only location where variables can be declared;

• The purpose of the **VAR block** is to define storage requirements, not to define data.

```
VAR
        myMessage : STRING;
```

• The general syntax for a variable declaration is:

<identifier>(,<identifier>,...) : <data type>;

```
jobName:STRING;
i,j,k:INTEGER;
```

Variables, and Constants

Constants

• The **CONST block** in the VectorScript is the only location where constants can be declared;

• Constants, unlike variables, do not require an explicit data type..

```
CONST
        kGREETING : ´Hello´;
```

• The general syntax for a variable declaration is:

<identifier> = <value>;

LOCAL_GREETING_FRENCH = ´Bonjour ´;

Data Types

Fundamental Data Types

- Numeric
- Text
- Other

Data Types

Numeric

- **INTEGER**
  -32767 to 32767
- **LONGINT**
  -2.147.183.647 to 2.147.183.647
- **REAL**
  1.9 x 10e-4951 to 1.1 x 10e4932

Data Types

Text

- **STRING**
  - up to 255 characters
  - ASCII character
- **CHAR**
  - a single ASCII character

Data Types

..Other

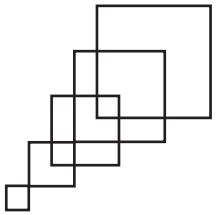
- **BOOLEAN**
  - TRUE or FALSE
- **HANDLE**
  - to store a **reference** to other VectorWorks data in memory.
- **VECTOR**
  - A VectorScript VECTOR consists of three **component** REAL values which can also be treated as a single unit value.
- **POINT**
  - to store the coordinates of a 2D point. It is a compound data type consisting of two **component** REAL values: **x** and **y**.
- **POINT3D**
  - to store the coordinates of a point in 3D space. It is a compound data type consisting of three **component** REAL values: **x**, **y** and **z**.
- **RGBColor**
  - The RGBCOLOR data type can store a color as three components: **red**, **green**,and **blue**. Each component is a LONGINT value.
- **NIL**

Repetition Statements

The FOR Statement

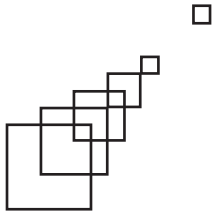
- **FOR ... TO... DO**

```
PROCEDURE LOOP;
VAR
        i : INTEGER;
BEGIN
        FOR i:=1 TO 5 DO rect(i,i,i*2,i*2);
END;
Run(LOOP);
```



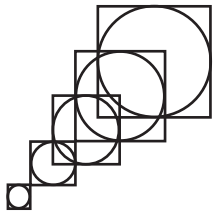
- **FOR ... DOWNTO... DO**

```
BEGIN
        FOR i:=1 DOWNTO -5 DO rect(i,i,i*2,i*2);
END;
```



- **FOR ... TO... DO BEGIN**

```
BEGIN
        FOR i:=1 TO 5 DO  BEGIN
                rect(i,i,i*2,i*2);
                oval(i,i,i*2,i*2);
                SysBeep;
        END;
END;
```



Repetition Statements

The WHILE Statement

- **WHILE ... DO**

```
PROCEDURE WhileLoop;
VAR
        h : HANDLE;
BEGIN
        h:= FActLayer;
        WHILE (h <> NIL) DO BEGIN
                SetSelect(h);
                h:=NextObj(h);
        END;
END;
Run(WhileLoop);
```

Repetition Statements

The REPEAT Statement

• **REPEAT ... UNTIL (...)**

```
PROCEDURE RepeatLoop;
VAR
        h : HANDLE;
BEGIN
        h:= FActLayer;
        REPEAT
                SetSelect(h);
                h:=NextObj(h);
        UNTIL (h=NIL);
END;
Run(RepeatLoop);
```

• Unlike the **WHILE** statement, however, the **REPEAT** statement evaluates the control expression after executing its controlled statement. This means that the controlled statement will **always execute at least once.**