

# Advanced Language Features

by Christian Schmitz

# Advanced Language Features

- Delegate
- Variant
- Pair
- Dictionary
- Class Interfaces
- Declare
- Operator Methods
- Static
- RBScript
- Exception

# Delegate

# Delegate

- Data Type for Method pointers
- C function pointers, but safe
- Contains reference to parent object used for creation.
- Declare in ContainerControl, Window, Class or Module like method

# Delegate Creation

- Create with Real Studio methods
  - AddressOf
  - WeakAddressOf
- Create with system function
  - Constructor(P as Ptr)

# Delegate Invoke

- Use Invoke Method
- Pass parameters declared before
- No optional parameters, please.
- `d.Invoke(x,y)`

# Delegate Sample

- Declare Delegate:
  - `Test(d as double) as Double`
- In Code:
  - `dim t as test = AddressOf Sqrt`
  - `MsgBox str(t.Invoke(25))`

# Delegate Usages

- With Declares
- Dictionary mapping name to method via delegate
- Passing callback function to methods



Variant

# Variant

- Variant is a container class
- Automatic data type conversion
- `dim v as variant = window1`  
`dim w as window = v`

# Variant

- Variant can store any data type
  - All objects
  - nil
  - Strings, Numbers, Structures
  - Arrays

# Explicit Conversion

- BooleanValue
- Int32Value
- SingleValue
- Int64Value
- StringValue
- IntegerValue
- UInt32Value
- ColorValue
- UInt64Value
- CurrencyValue
- ObjectValue
- DateValue
- DoubleValue
- PtrValue

# Variant Types

- Type Property or `VarType()`
- Constants in Variant Class
- `if VarType(v) = Variant.TypeArray + Variant.TypeInteger then  
    dim values() as integer = v  
end if`
- Avoids `TypeMismatchException`

# Variant Types

- Avoids `TypeMismatchException`
  - `dim w as window = window1`  
`dim v as Variant = w`  
`dim n as integer = v`
  - `dim v as Variant = 1.0`  
`dim d as date = v`
- Unless matching `Operator_Convert` method exists.

# Variant Conversions

- Convert Color ↔ String

- `dim c as color = &c112233`  
`dim v as Variant = c`  
`dim s as string = v // &h00112233`

- Convert Boolean ↔ String

- `dim s as string = "true"`  
`dim v as Variant = s`  
`dim b as Boolean = v // true`

# Variant Usages

- Tag Properties
  - e.g. RowTag, CellTag and ColumnTag in Listbox
- Generic classes like Dictionary & Pair
- Used for AddressBookContact/Group properties



# Class Interface

# Interface

- Like a class without code
- Interface defines methods
- Classes conform to interfaces
- Workaround for missing multiple inheritance

# Interfaces in Framework

- Readable
- Writeable
- PreparedStatement
- DataSet (for Reports)

# Readable Interface

- Implemented by
  - BinaryStream
  - IPCSocket & TCPSocket
  - Serial
  - Stdin
  - TextInputStream

# Interface Example

- Sub Write(r as Writeable, s as String)  
    r.Write(s)  
End Sub
- Write(mySocket, „Hello“)
- Write(mySerial, „Hello“)
- Write(myStream, „Hello“)

Pair

# Pair Class

```
class Pair
  // Properties
  Left as Variant
  Right as Variant
  // Methods
  Constructor(left as Variant,
              right as Variant)
end class
```

# Pair Syntax

- `:` operator for easy Pair creation
- Same
  - `Dim p as Pair = 1 : 2`
  - `Dim p as new Pair(1,2)`



# Pair Usage

- Return multiple results:
  - `return value:status`
- Create Dictionary with Pairs
  - `dim d as new Dictionary("left" : 0, "top" : 10)`

# Dictionary

# Dictionary

- Container class for Key → Value
- Fast lookups due hash table
- Stores variant
- Good for lookup tables

# Dictionary

- Create dictionary
- `dim d as new Dictionary("left" : 0, "top" : 10)`
- `dim d as new Dictionary`  
`d.Value(key) = value`

# Dictionary

- Add or Replace Value
  - `d.Value(key) = value`
- Remove value
  - `d.remove(key)`
- Remove all
  - `d.clear`

# Dictionary

- Query all keys
  - `d.keys()` as variant
  - `d.count` and `d.key(index)`
- Query all values
  - `d.values()` as variant

# Dictionary

- Lookup with exception
  - `value = d.value(key)`
  - Can raise `KeyNotFoundException`
- Lookup without exception
  - `value = d.Lookup(key, defaultValue)`
- Test key
  - `HasKey(key)` as boolean

# Dictionary

- Loop over all keys and values
- for each key as variant in d.keys  
  dim value as variant = d.value(key)  
  msgbox key + " → " + value  
  next
- Or loop over all values with d.values



# Declare

# Declare

- Call functions in shared libraries without plug-in
- Pick function details from C Headers

# Declare

- Translate types from C to Real Studio
- `const char*` → `CString`  
`const wchar_t*` → `WString`
- Must match parameters exactly
- Must know path to library
- Library can be constant for different names on different platforms.

# Declare

- Declare Function <name> Lib <library> Alias <aliasname> (<parameter>) as <returntype>
- Declare Sub <name> Lib <library> Alias <aliasname> (<parameter>)

# Declare

- C function: `pid_t getpid(void);`
- Declare Function `getpid` Lib "LibC" () as Integer
- `msgbox str(getpid)`

# Operator Methods

# Operators

- Overload operators for classes
- Using = for compare calls Operator\_Compare

# Operators

- Operator\_Add
- Operator\_Subtract
- Operator\_And
- Operator\_Or
- Operator\_Divide
- Operator\_Modulo
- and more...
- Operator\_Convert
- Operator\_Power
- Operator\_Negate
- Operator\_Lookup
- Operator\_Redim
- Operator\_Compare



# Operator\_Convert

- Vector class with x, y as double:
- Function Operator\_Convert() As string  
Return str(x)+"/"+str(y)  
End Function
- dim v as new vector(5,6)  
msgbox v
- Shows „5/6“

# Operator\_Add

- Function Operator\_Add(v as Vector) As vector  
return new Vector(v.x+x, v.y+y)  
End Function
- dim v as new Vector(5,6)  
dim w as new Vector(3,1)  
MsgBox v+w
- Shows „8/7“

# Operator\_Compare

```
Function Operator_Compare(v as Vector) As integer
  if v.x = x and v.y = y then
    Return 0 // equal
  elseif v.x*v.x + v.y*v.y > x*x + y*y then
    Return -1 // smaller
  else
    return 1 // bigger
  end if
End Function
```

# Operator\_Compare

```
dim v as new Vector(5,6)
dim w as new Vector(3,1)
if v<w then
    MsgBox "smaller"
else
    MsgBox "bigger or equal"
end if
```

# Operator\_Compare

- Using = for comparison can call Operator\_Compare
- If x = nil then // calls compare
- If x is nil then // compares pointers
- Handle nil in Operator\_Compare!

Static

# Static

- Declare global variable in a method
- Only visible locally where declared
- Example:
  - static cache as new Dictionary
  - static counter as integer = 0

# R.BScript



# RBScript

- Execute RB code in your application at runtime
- Evaluate expressions
- Provide own scripting language inside app
- Fast and already using LLVM

# RBScript Examples

- Example:
- `RbScript1.Source = "print str("+TextField1.text+")"`  
`RbScript1.Run`
- `Sub RbScript1. Print(msg As String)`  
    `label1.text = msg`  
`End Sub`
- Evaluates formula in textfield and writes result to label.

# Exception

# Exception

- Structural error handling for fatal errors
- One method raises Exception
- Other method catches it
- Unhandled Exception Events
- All Exceptions are subclasses from RuntimeException

# Built-in Exception Classes

- NilObjectException
- OutOfBoundsException
- FunctionNotSupportedException
- IllegalCastException
- StackOverflowException
- IOException

# Raise Exception

- Raise a new exception
- ```
dim n as new KeyNotFoundException  
n.message = key+" not found"  
raise n
```

# Catch Exception

- try catch finally
- exception on method end
- unhandled exception handler

# Try command

- try

```
    dim b as binarystream = binarystream.open(f)
catch r as IOException
    msgbox „Can't open file“
finally
    // cleanup
end try
```
- Useful for local catching of expected exceptions like IOExceptions



# Try command

- `dim b as binarystream = binarystream.create(f)`  
`b.write „Hello World“`  
`b.close`
- `exception i as IOException`  
`msgbox „Failed to create file.“`
- Useful for catching all/some events in a method

# Unhandled Exceptions

- Log unhandled exceptions! Report them to developer!
- in UnhandledException event in App and Session classes
- If you do nothing your app quits

# Analyse Exceptions

- Name of exception
  - `Introspection.GetType(ex).name`
- Stack
  - `msgbox join(ex.stack, endofline)`
- Error Message and Error Code
  - `msgbox "Error "+str(ex.errorCode)+"": "+ex.message`

Questions?