

Manual de Referencias

Sora Script

0.1.1368

Copyright (C) 2008 Osmely Fernández Llitas, Ronny Rodríguez Lloró, José Seidel L. García, Aleya Valdés González, Eliezer de Armas Pestana, Johan Vivas, Luis Prada, Alejandra Martínez.

Instituto Superior Pedagógico "Rubén Martínez Villena" (MINED)(Centro de Estudios de Software Educativos); Ministerio del Poder Popular para la Educación (MPPE) (Fundación Bolivariana de Informática y Telemática FUNDABIT). Registro: 2821-2008.

Prefacio

El desarrollo de un sistema de autor que brinde las herramientas suficientes para la elaboración de materiales digitales educativos por parte de un número amplio y diverso de usuarios, es el objetivo fundamental perseguido por el equipo de desarrollo que estuvo a cargo de la programación del presente software. La facilidad, flexibilidad y estabilidad de dicho sistema, son cualidades que hemos tratado de reflejar en su diseño funcional e implementación. Las futuras versiones mostrarán además el siempre presente intento de satisfacer las necesidades de los usuarios y las nuestras como programadores.

En este material se pretende hacer referencia a los comandos, estructuras, etc., que pueden ser empleados durante la programación escrita. Puesto que actualmente la herramienta emplea el intérprete de wxBasic, a la gran mayoría de las funciones se les ha mantenido la sintaxis original de este lenguaje, mientras que, además se incluyen un grupo importante de sentencias propias de la herramienta, que fueron adicionadas al intérprete para el manejo de los objetos y el entorno, fundamentalmente. A esta fusión entre los comandos de wxBasic y los propios del proyector de HAEduc es a lo que llamamos Sora Script.

Tanto las referencias que se brindan como los ejemplos recogidos en este manual han sido enfocados suponiendo que el lector tiene conocimientos básicos de la programación en Basic.

La documentación que se brinda por parte del autor del intérprete de wxBasic fue un material del que se extrajo una gran parte de lo que aquí se explica sobre wxBasic.

IMPORTANTE:

Puesto que el compilador de wxBasic ha sido adaptado para su empleo en esta Herramienta de Autor, advertimos que el uso de comandos que no estén recogidos en este manual de referencias puede ocasionar comportamientos impredecibles en las aplicaciones programadas bajo este entorno.

Agradecimientos

Agradezco la importante participación de un grupo de compañeros y amigos que sin ellos este producto no tendría la calidad que posee:

La oportuna colaboración del licenciado Amaury Pérez Torres y el ingeniero Andreys Reytor López por su labor para la solución de importantes dificultades.

El aporte en el diseño gráfico de Yoenis León Mesa.

Las oportunas sugerencias y señalamientos del doctor Carlos E. Expósito Ricardo y del Master en Ciencias Cesar Labañino Rizo, poniendo siempre a nuestra disposición sus experiencias en el área de la informática educativa.

También a todos los que han invertido un segundo de su tiempo en este importante proyecto.

Índice de contenido

Sobre el lenguaje y la Herramienta.....	7
¿Qué es Sora Script?.....	7
Fundamentos de WxBasic.....	7
Colecciones.....	14
Métodos de colecciones.....	14
Array.....	15
Table.....	15
List.....	16
Comentarios.....	18
Ámbito de variables.....	18
Constantes.....	20
Estructuras cíclicas.....	20
FOR...NEXT.....	20
WHILE ... END WHILE.....	21
FOR EACH...END FOR.....	21
Estructuras condicionales.....	22
IF ... ELSEIF ... ELSE ... END IF.....	22
Select case.....	23
Procedimientos sub.....	24
Funciones.....	24
Clases.....	26
Clases Internas de Sora Script.....	28
Clase ssRect.....	28
Clase ssPunto.....	29
Clase ssColor.....	29
Clase ssCC.....	30
Clase ssDim.....	34
Clase ssFuente.....	35
Clase ssNullImagenBase.....	37
Clase ssImagenBase.....	37
Clase ssSonido.....	39
Trabajo con ficheros.....	40
SHARED.....	41
STATIC.....	41
PRINT #.....	42
Otras funciones para el trabajo con ficheros y directorios.....	42
Otras Funciones de Sora Script.....	43
Funciones de navegación entre páginas.....	43
Otras funciones.....	44
Diálogos comunes.....	47
clase ssDialogoColor.....	48
clase ssDialogoFichero.....	48
clase ssDialogoDirectorio.....	49
clase ssDialogoTexto.....	50
clase ssDialogoFuente.....	50
clase ssDialogoMensaje.....	51
Los Objetos básicos del proyector.....	52
Clase SSOBJETO.....	53

Clase SSTabla.....	54
Clase SSEtiqueta.....	58
Clase SSPoligono.....	58
Clase SSCtexto.....	60
Clase SSImagen.....	60
Clase SSHtml.....	61
Clase SSLista.....	63
Clase SSReloj.....	64
Clase SSVideo.....	65
Clase SSTexto.....	66
Bases de datos.....	69
Clase ssBaseDatos.....	69
Clase ssResultado.....	69
Captura de eventos.....	71
Los objetos básicos del proyector y las clases.....	81
El Runtime.....	86
Palabras reservadas.....	87
Operadores.....	88
Operadores de asignación.....	88
Funciones nativas del lenguaje.....	88

Sobre el lenguaje y la Herramienta

HAEduc es una Herramienta libre, multiplataforma (Windows y Linux) para el desarrollo de aplicaciones educativas. Cuenta con los elementos necesarios (objetos de usuario) y un lenguaje (Sora Script) suficientemente potente para crear software de este tipo. Gracias a las herramientas que brinda permite que un amplio grupo de usuarios la empleen al poseer los elementos necesarios para el trabajo desde usuario inexpertos en el empleo de herramientas de autor hasta expertos en esta materia.

¿Qué es Sora Script?

El runtime de HAEduc emplea un intérprete de un lenguaje muy cercano al Basic, el intérprete de wxBasic, cuyo autor es David Cuny. Dicho intérprete es libre (licencia LGPL) y se puede descargar en <http://wxBasic.sourceforge.net>. Los autores de HAEduc emplearon este intérprete libre y a partir de modificaciones realizadas en el mismo, para adaptarlo a la metáfora que propone el sistema y con la adición de un proyector y objetos visuales y los métodos para su manejo, surge Sora Script que mantiene como esencia a wxBasic. Por ello este manual de referencias hará especial énfasis en la programación con este lenguaje (wxBasic) y también mostrará los nuevos elementos para la explotación de la metáfora que plantea HAEduc y Sora Script.

Fundamentos de WxBasic

Las principales características del lenguaje son:

- Es libre.
- Multiplataforma (Windows y Linux).
- Fácil de enseñar y aprender, basado en Basic.
- Interpretado.
- El runtime ocupa poco espacio en disco.

En su sintaxis se revelan elementos de C, QBasic, Lua, Python y VB.NET. Si bien no es un lenguaje muy potente, los autores de HAEduc lo consideran suficiente para el desarrollo de recursos educativos y aplicaciones no muy complejas. Unido al hecho de que wxBasic es más flexible que el Basic comúnmente conocido y ha sido diseñado con muchas mejoras. Si a esto le sumamos las posibilidades gráficas del entorno de HAEduc, se puede disfrutar de un ambiente de desarrollo estable, acorde a las necesidades de un amplio grupo de usuarios.

¿Cómo wxBasic se diferencia de otros Basic?

Los principales elementos que diferencian a wxBasic del Basic son los siguientes:

- **Los tipos de datos simples no son pasados por referencia**

Algunos tipos de datos (como el tipo Array y Objects) son pasados a las rutinas por referencia, pero los tipos de datos simples (como Number y String) no lo son. Esto significa que si un parámetro de un tipo de dato simple, cambia en el interior de una rutina, no cambiará el valor de la variable empleada para la llamada. Por ejemplo:

```
Sub foo( a )
  a = a + 10
  ssMensaje ("En foo(), a = " & a)
End Sub

b = 100
ssMensaje ("Antes de llamar a foo(), b = " & b)
foo( b )
ssMensaje ("después de llamar a foo(), b = " & b)
```

En algunas versiones de Basic, cambiando el valor de **a** debe cambiar el valor de **b**. Este no es el caso de wxBasic. Si se desea cambiar el valor de **b**, necesita asignarle el valor de retorno.

```
Function foo( a )
  a = a + 10
  ssMensaje ("En foo(), a = " & a)
End Function

b = 100
ssMensaje ("Antes de llamar a foo(), b = " & b)
b = foo( b )
ssMensaje ("después de llamar a foo(), b = " & b)
```

- **Las funciones devuelven valores mediante la palabra reservada Return**

Para devolver un valor desde una función en wxBasic, se necesita usar la palabra reservada **Return**. Esta funciona como en **C**: se sale de la función en este punto, y devuelve el valor que sigue a dicha palabra:

```
Function foo( a )
  Return a * 10
End Function
```

Si se quiere continuar procesando dentro de la función, puede emplearse una variable temporal:

```
Function foo( a )
```

```
Dim tmp = a * 10
' hacer algo más
Return tmp
End Function
```

- **Las funciones pueden devolver múltiples valores**

Al igual que **Lua** y **Python**, wxBasic permite funciones que devuelvan más de un valor. Por ejemplo:

```
Function doubleTheValue( a )
' Return double the value and original value
Return a*2, a
End Function
```

```
newValue, oldValue = doubleTheValue( 120 )
```

wxBasic hace un chequeo de la cantidad de argumentos de ambos lados de la expresión, de forma tal que le asignará a los miembros izquierdos las expresiones de la derecha teniendo en cuenta el orden respectivo. Si el número de argumentos a la izquierda es mayor que el de la derecha, se le asigna el valor **Nothing** al argumento restante. Por otra parte si el número de expresiones o valores a asignar es mayor que el de argumentos a la izquierda, estos serán ignorados por el compilador, por ejemplo:

```
a, b = 10, 20, 30
```

En este caso, a **a** se le asigna el valor 10, y a **b** el valor 20. Como no hay variable para recibir el valor 30, este será ignorado por el compilador. Similarmente:

```
a, b = 10
```

En este caso, **a** tomará el valor 10, y **b** el valor **Nothing**.

Si una función no devuelve explícitamente un valor, esta devolverá por defecto el valor **Nothing**:

```
Function doNothing()
End Function
```

```
a = doNothing()
```

Esto hace que a la variable **a** se le asigne el valor **Nothing**.

- **Las funciones pueden tener parámetros opcionales**

Se puede especificar que una rutina puede tener parámetros opcionales. Si la rutina es llamada sin estos parámetros, ellos tomarán los valores por defecto. Por ejemplo:

```
Sub hasOptionalParms( a, b, c = 10, d = 100 )
  ssMensaje (a & " " & b & " " & c & " " & d)
End Sub
```

- **Las funciones pueden tener un número variable de parámetros**

Se puede especificar que una rutina tenga un número de variable de parámetros. Similar al comando **printf** de **C**:

```
Sub hasVariableParms( a, b, ... )
```

- **Declaración dinámica de variables**

En wxBasic, no es necesaria la declaración explícita de variables, porque son creadas “al vuelo” por el compilador en el momento de usarlas:

```
aString = "esto es una cadena"
aNumber = 123
```

Las variables declaradas fuera de las rutinas son automáticamente de ámbito global:

```
globalVar = "Soy una variable global"
```

```
Sub foo()
  localVar = "Soy una variable local"

  ' Referencia a la variable global globalVar
  ssMensaje (globalVar)

  ' Referencia a la variable local localVar
  ssMensaje (localVar)

End Sub
```

Como conveniente es esta opción, lo es más si se establece como obligatoria la declaración explícita de variables. Esto podrá hacerse estableciendo la siguiente declaración:

```
Option Explicit
```

Con **Option Explicit** habilitado, se vuelve obligatoria la declaración explícita de las variables antes de usarlas:

```
Dim globalVar = "Soy una variable global"

Sub foo()
  Dim localVar = "Soy una variable local"
```

' Referencia a la variable global globalVar
 ssMensaje (globalVar)

' Referencia a la variable local localVar
 ssMensaje (localVar)

End Sub

- **No es sensible a mayúsculas y minúsculas**

El compilador de wxBasic no detecta diferencia entre mayúsculas y minúsculas, por tanto los siguientes casos significan lo mismo:

A=22 es lo mismo que **a=22**

A=Sin(90) es lo mismo que **a=SIN(90)**

ssMensaje ("hola mundo") es lo mismo que **ssMensaje ("hola mundo")**

ssMensaje (True) es lo mismo que **ssMensaje (TRUE)**

- **Tipos de datos**

Todos los valores en wxBasic son almacenados por defecto como **Variants**. Sin embargo se pueden definir las variables con los siguientes tipos:

- **Variant**
- **Integer**
- **Number**
- **String**
- **DateTime**
- **Object**
- **Routine**
- **Array**
- **Table**
- **List**

Existen métodos comunes para algunos de estos tipos de datos:

```
integer = GetType(variant)
string = ToString(variant)
integer=variant.Equal( OtroValor)
integer=variant.Compare( OtroValor)
clone = variant.Clone()
```

Nothing

Nothing es el equivalente en otros lenguajes a **NIL** o **NULL**. Es equivalente a False en una prueba de veracidad. Todas las variables tienen este valor por defecto. Se puede emplear este valor para probar el contenido de una variable:

```
If a = Nothing Then
  ssMensaje ("a = Nothing")
End If
```

Este implementa los siguientes métodos:

```
isNumeric
fromChar
toChar
isTrue
```

Integer

Integer es un entero. En estas variables se almacenan valores enteros. Todas las operaciones matemáticas son internamente hechas con **double**, incluso cuando ambos tipos son declarados como **Integer**.

Number

Number es un número con punto flotante, almacena como un double de C.

String

String es una cadena de caracteres alfanuméricos. Cuando se hace una prueba de verdad, las cadenas vacías devuelven un valor False, y las no vacías devuelven **True**.

Se pueden incluir caracteres especiales dentro de la cadena usando el backslash (\):

```
\n  nueva línea
\r  retorno de carro
\t  tab
\'  comilla simple
\"  comilla doble
\\  backslash
```

Por ejemplo:

```
stringConDosLineas = "Esta es la primera línea\nEsta es la segunda"
```

Se puede tratar una cadena como si esta fuera una colección. Se puede iterar dentro de la cadena. Por ejemplo:

```
For Each posicion, letra In "hola, mundo"
  ssMensaje ( "Posición= " & posicion & " letra= " & letra)
End For
```

Se puede además usar índices en las cadenas como en un array, por ejemplo:

```
a= "Hello, World"[8]
```

Le asignará **"W"** a la variable **a**, y:

```
a= "Hello, World"[1:5]
```

Le asignará **"Hello"** a la variable **a**.

DateTime

DateTime representa fecha y hora. Este es normalmente creado asignándole una cadena con formato de fecha:

```
Dim myDate = "Jan 12, 2003"
```

Hasta ahora sólo se admite en trabajo con fechas en Inglés.

Se pueden extraer las partes de la fecha llamando a varios métodos del lenguaje:

```
Year(myDate)  
Month(myDate)  
MonthName(myDate)  
Weekday(myDate)  
Day(myDate)  
DayName(myDate)  
Hour(myDate)  
Minute(myDate)  
Second(myDate)
```

El empleo del método Now() permite obtener la fecha actual.

Object

Cualquier dato declarado como una clase (interna o de usuario) es almacenada como un tipo de dato **Objects**. Los Objects son pasados por referencia, no por valor. Para crear una copia de un objeto se tiene que emplear el método **Clone**:

```
miCopia = Clone(EIObjeto)
```

Rutinas

Una rutina (**Routine**) es un método o rutina definida por el usuario. Este es actualmente un tipo de dato interno, no debe emplearse para la versión actual del compilador.

Colecciones

Una colección es un agrupamiento de datos **Variant** a los que se puede tener acceso mediante un valor clave. Por ejemplo, **Array**, **Table**, y **List** son todos, tipos de datos de colección. Estas por tanto tienen operaciones comunes (en general) a las colecciones. Se puede acceder a un elemento de la colección especificando el índice. Por ejemplo, esta es una colección **Table**:

```
Dim misgatos = { }
```

Índices

Una tabla (**Table**) podría ser esta:

```
misgatos [ 1 ] = "Chester"
misgatos [ 2 ] = "Charlotte"
misgatos [ 3 ] = "Chloe"
```

Si se imprime el valor de la tabla:

```
ssMensaje (misgatos)
```

Se obtendrá: **{ 1:"Chester", 2:"Charlotte", 3:"Chloe" }**

Podremos acceder a los elementos en una colección especificando la colección y el índice:

```
ssMensaje (misgatos [ 3 ])
```

Imprimirá: **"Chloe"**.

Métodos de colecciones

En adición a los métodos implementados por los **Variants**, las colecciones implementan uno o más de estos métodos:

```
colección.Append( valor )
colección.Prepend( valor )
colección.InsertAt( key, valor )
colección.Slice( startSlice, endSlice )
integer = colección.Count
```

También estos métodos son generales del script y se pueden emplear de la siguiente forma:

```
Append( colección, valor )
Prepend(colección, valor )
InsertAt( colección, key, valor )
Slice( colección, startSlice, endSlice )
integer = Count(colección)
```

Array

Un Array es una colección de tamaño estático. Como las demás colecciones, es pasada por referencia. Se emplea el corchete para distinguirlos de una llamada a rutina. Veamos una declaración simple de un array:

```
Dim miArray[10]
```

Si el límite inferior del rango no es declarado, este se toma como cero. Un array puede tener hasta cuatro dimensiones:

```
Dim miArray[5 To 20, -10 To 10]
```

Un array que no haya sido declarado como un tipo de dato, sus valores se inicializan a cero. Es posible declarar un valor inicial a un array para sus elementos:

```
Dim miArray[10] = Nothing
```

Se puede especificar un tipo de dato:

```
Dim miArray[10] As Number
```

También es posible hacer ambas cosas:

```
Dim miArray[10] As Integer = 30
```

Table

Una tabla es una colección no ordenada de tamaño dinámico. Como todas las colecciones, es pasada por referencia. Se puede declarar una colección empleando las llaves ({}). Así se podría declarar una tabla vacía:

```
Dim miTabla = {}
```

Una colección es indexada mediante valores, ya sea numéricos (**Integer**) o de cadena (**String**). (Los numéricos son automáticamente convertidos a enteros). Por ejemplo:

```
Dim misgatos = { 1:"Chester", 2:"Charlotte", 3:"Chloe", 4:"Julius" }
```

es lo mismo que escribir:

```
Dim misgatos = {}
misgatos [1] = "Chester"
misgatos [2] = "Charlotte"
misgatos [3] = "Chloe"
misgatos [4] = "Julius"
```

Observe que si el elemento no existe en la lista, wxBasic lo insertará como un nuevo elemento de la colección.

Si no se especifica el índice en la declaración, se asignarán secuencialmente los valores. Por tanto lo anterior también podría escribirse como:

```
Dim misgatos = { "Chester", "Charlotte", "Chloe", "Julius" }
```

Se puede acceder a un elemento especificando el índice:

```
ssMensaje(misgatos [1])
```

Si el índice no existe en la tabla, se devolverá un valor **Nothing**. Si la lista contiene otra lista, se puede acceder a esta como en un array:

```
Dim misgatos = { }
mischicos[ "Chester", "comida favorita" ] = "pescado"
```

Igualmente se podría escribir:

```
Dim misgatos = { }
mischicos [ "Chester" ] = { "comida favorita":"pescado" }
```

que es además igual que:

```
Dim misgatos = { "Chester":{ "comida favorita":"pescado" } }
```

Y este puede ser accedido similarmente:

```
ssMensaje(misgatos [ "Chester", " comida favorita " ])
```

Imprime: **"pescado"**

Como con las demás colecciones, se puede iterar dentro de esta con la estructura repetitiva **For Each**:

```
For Each indice, nombre In misgatos
  ssMensaje(indice, nombre)
End For
```

Es posible remover elementos de la lista mediante el valor del elemento:

```
misgatos.Remove( "Julius" )
```

Para saber de la existencia de un índice en una tabla emplee en método HasKey. Véase método HasKey.

List

Una lista es una colección ordenada de tamaño dinámico. Como las

demás colecciones esta es pasada por referencia. Se puede declarar una tabla empleando los corchetes ([]). El siguiente ejemplo muestra una lista vacía:

```
Dim milista = []
```

La construcción de una tabla es similar a la de una lista, excepto que los índices son implícitos. Basados en el orden:

```
Dim misgatos = [ "Chester", "Charlotte", "Chloe", "Julius" ]
```

Se puede adicionar elementos a una lista con **Append**, que los colocará al final de la lista:

```
Dim misgatos = []
misgatos.Append( "Chester" )
misgatos.Append( "Charlotte" )
misgatos.Append( "Chloe" )
misgatos.Append( "Julius" )
```

o con **Prepend**, que los adiciona al inicio:

```
Dim misgatos = []
misgatos.Prepend( "Julius" )
misgatos.Prepend( "Chloe" )
misgatos.Prepend( "Charlotte" )
misgatos.Prepend( "Chester" )
```

o mediante **Insert**, especificando la posición a insertar el elemento:

```
Dim misgatos = []
misgatos.Insert( 1, "Julius" )
misgatos.Insert( 2, "Chloe" )
misgatos.Insert( 3, "Charlotte" )
misgatos.Insert( 4, "Chester" )
```

Los elementos pueden ser accedidos mediante los índices, comenzando por 1.

El acceso a un elemento que no exista provocará un error.

```
ssMensaje(misgatos [3])
```

Es posible acceder además a un fragmento de la lista:

```
ssMensaje(misgatos [2:3])
```

Igual resultado se obtendrá empleando el método `slice`

```
ssMensaje(Slice(misgatos,2,3))
```

Comentarios

Los comentarios en Sora Script pueden establecerse en una línea simple o en forma de bloques al estilo del lenguaje C.

El comentario para una sola línea se establece mediante el símbolo de comilla simple (') o al estilo de C con dos slash (//). Ambos símbolos establecen todo el texto hacia la derecha a partir de ellos como comentarios hasta el fin de línea:

```
'esto es un comentario  
//esto también
```

Los bloques de comentarios se inician con los símbolos /* y se cierran con */

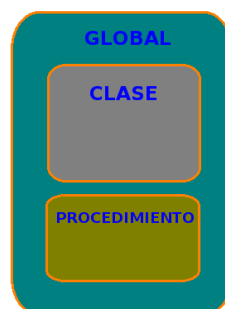
```
/* Este es otro comentario  
en forma de bloque */
```

Ámbito de variables

Los ámbitos de variables dependen del modo en que se emplee en runtime de HAEduc (ver el apartado *El Runtime*).

Si el runtime se emplea en modo script:

En el modo script existen los ámbitos global, de clase y local de procedimientos (sub y function). El ámbito global es el que está fuera de los procedimientos, las funciones y clases, es el ámbito más externo y superior.



En cuanto el compilador encuentra una declaración (explícita o implícita) de variable, este verifica si no existe en el ámbito actual, en este caso la crea, sino genera un error.

Si existen dos variables de igual nombre que pertenecen a ámbitos distintos que se solapan, tendrá prioridad la variable local ante la global:

```
' Esta es una declaración en ámbito global
Dim myVar = "variable global"
Function MiRutina()
' Esta está declarada a nivel de rutina, local
Dim myVar = "variable de la rutina"
' Se usa la versión local de myVar
ssMensaje(myVar)
End Function
```

Si no se encuentra una variable en el ámbito actual se continuará la búsqueda en el ámbito global:

```
' Esta es una declaración global
Dim myVar = "variable de módulo"

Function MiRutina()
' Se usa la variable global
ssMensaje(myVar)
End Function
```

Si necesita emplear más de una sentencia en una misma línea podrá emplear los dos puntos (:) para separarlas:

```
A=22: b= "texto": ssMensaje(A)
```

Nota: En modo script las clases deben estar situadas al inicio del primer fichero de script, evitando colocar procedimientos antes de las clases.

Si el runtime se emplea en modo de proyecto:

En modo de proyecto los ámbitos de clase y local de procedimientos (sub y function) se mantienen pero el global ahora desaparece para dar lugar a los ámbitos de Libro y de Página. Como cada proyecto sólo posee un libro, este constituye el ámbito superior, mientras que al existir un número indeterminado de páginas cada una de ellas constituye un ámbito en el que coexisten variables y procedimientos.



Nota: En modo de proyecto las clases deben estar situadas el ámbito de Libro.

Constantes

Sintaxis:

CONST nombreconstante = expresión {, nombreconstante = expresión }

Declara una variable como constante. Las constantes son similares a las variables, sólo que no se les puede modificar su valor.

```
' Declaro el nombre de mi programa
CONST NombreProg = "Mi Programa", Version = "1.0"
```

Estructuras cíclicas

wxBasic permite crear estructuras repetitivas de varias formas:

Las clásicas de Basic:

```
For ... Next
While ... Wend
```

O las inspiradas en Algol:

```
For ... End For
While ... End While
```

FOR...NEXT

Sintaxis:

```
FOR variable = startExpr TO endExpr {STEP stepExpr}
    [CONTINUE]
    [BREAK]
    [EXIT FOR]
    { statement }
ELSE
    { statement }
END FOR | NEXT {variable}
```

Este es un ejemplo de un ciclo que imprime un número entre 1 y 10

```
' Ciclo de 1 a 10
FOR i = 1 TO 10
    ssMensaje(i)
NEXT
```

Si el ciclo no es detenido por el comando BREAK, se ejecutará lo que se encuentra después del ELSE

```
FOR i = 1 to LENGTH( list )
    IF list[i] = someValue THEN
        ssMensaje("Encontrado!")
        BREAK
    END IF
ELSE
    ssMensaje("No fue encontrado")
END FOR
```

Continue provoca un salto al inicio del bucle donde se evalúa nuevamente la **expresión**.

WHILE ... END WHILE

```
WHILE expresión
    [ BREAK ]
    [ CONTINUE ]
    [ EXIT WHILE ]
    { instrucciones }
[ELSE
    { instrucciones } ]
END WHILE | WEND
```

Repite un conjunto de sentencias mientras la **expresión** sea verdadera. El comando Break termina inmediatamente el ciclo y **Continue** provoca un salto al inicio del bucle donde se evalúa nuevamente la **expresión**.

Por ejemplo:

```
' buscar un texto en un fichero
WHILE NOT EOF()
    INPUT #1, texto
    IF instr( texto, textobuscado ) THEN
        ssMensaje("Texto encontrado")
        BREAK
    END IF
ELSE
    ssMensaje("Texto no encontrado")
END WHILE
```

FOR EACH...END FOR

```
FOR EACH variable {, variable} IN expresión
```

```

    [CONTINUE]
    [BREAK]
    [EXIT FOR]
    { instrucciones }
ELSE
    { instrucciones }
END FOR

```

Recorre una colección (table o list). Si sólo se especifica una variable de ciclo esta contendrá el índice en cada ciclo, por otra parte si se emplean dos variables, entonces contendrán el índice y el valor de este elemento de la colección. CONTINUE, BREAK y ELSE se comportan de la misma forma que en el ciclo FOR.

```

'Imprime el valor y el índice de una lista
FOR EACH key, value IN list
    ssMensaje(key & " " & value)
END FOR

```

Estructuras condicionales

IF ... ELSEIF ... ELSE ... END IF

```

IF expresión THEN
    { instrucciones }
{ ELSEIF expr THEN
    { instrucciones } }
[ ELSE
    { instrucciones } ]
END IF

```

Ejecuta un bloque de instrucciones dependiendo del valor de una expresión evaluada en cada ciclo.

Sintaxis:

```

If condición then
{estas_instrucciones}
else
{otras_instrucciones}
End if

```

por ejemplo:

```

if a=2 then
b=0
else
b=1
end if

```

'Se traduce: si a es igual a 2 entonces b tomará valor 0, sino se le asignará valor 1

Las condicionales también pueden escribirse en una sola línea aunque en este caso no podrá emplearse la sentencia **else**.

If condición then instrucciones

Comentarios

De los dos formatos antes expuestos (una sola línea o de bloque) emplee la forma de una línea para condicionales cortas, mientras que el formato de bloque le brinda más facilidades para la programación de condicionales más complejas y para la lectura e interpretación posterior del código.

Por otra parte mediante el empleo del formato de una sola línea y de los dos puntos se podrán ejecutar varias instrucciones en una sola condicional.

```
If S=1 then S=S+1: D=D+1: J=J+1
```

Una expresión numérica es tratada como verdadera si su resultado es distinto de cero. En general wxBasic se encargará de realizar las conversiones numéricas necesarias, siempre que le sea posible. Las excepciones son el tipo de dato Nothing (este siempre es falso) y el string (cadena de caracteres). Una cadena de caracteres vacía es tratada como *false*, de lo contrario será *true*.

Las siguientes expresiones siempre serán evaluadas como *true*:

```
IF "0" THEN
    ssMensaje( "Esto es verdadero porque el string no está vacío")
END IF
```

Si se desea hacer una comparación numérica con un string hay que asegurarse de realizar la conversión primero:

```
IF VALUE("0") THEN
    ssMensaje("esto es falso, porque el numero 0 es tratado como false")
END IF
```

Usar ELSE IF en lugar de ELSEIF también se acepta.

```
IF a = 1 THEN
    ssMensaje("One")
ELSE IF a = 1 THEN
    ssMensaje("Two")
ELSEIF a = 3 THEN
    ssMensaje("Three")
ELSE
    ssMensaje("Too big!")
END IF
```

Select case

SELECT CASE ... END SELECT

```
SELECT CASE expresión
{ CASE caseTest {, caseTest }
    { instrucciones } }
```

```
[CASE ELSE
    { instrucciones } ]
END SELECT
```

Permite ejecutar uno de varios bloques de instrucciones. Al igual que en C, sólo un caso será ejecutado. Se pueden utilizar expresiones múltiples o intervalos en cada cláusula mediante las siguientes sintaxis de operadores:

```
IS = | <> | < | > | < | >= expr
expr TO expr
expr
```

por ejemplo:

```
SELECT CASE a
CASE 1, 3
    ssMensaje("el valor es 1, o 3")

CASE 4 TO 6, 8
    ssMensaje("el valor es 4, 5, 6, o 8")

CASE IS < 12
    ssMensaje("el valor es menor que 12")

CASE ELSE
    ssMensaje("el valor el algún otro, distinto a los anteriores")

END SELECT
```

Nota importante: No deje líneas en blanco entre el Select case y el primer case.

Procedimientos sub

```
SUB ... END SUB
    SUB name ( [arg [= expr] {, arg [= expr]} ] [, ...] )
        [ DIM variable {, variable} ] ]
        [ STATIC variable {, variable} ] ]
        [ SHARED variable {, variable} ] ]
        [ RETURN ] ]
        [ EXIT SUB ] ]
        { instrucciones }
    END SUB
```

Un procedimiento Sub es un bloque de código que puede ser ejecutado a voluntad, en el momento que se requiera. Es esencialmente como una función, sólo que este no devuelve valores.

Los procedimientos permiten seccionar el código de un programa para ganar en claridad, eficiencia y robustez.

Funciones

```
FUNCTION ... END FUNCTION
    FUNCTION name ( [arg [= expr]{,arg [= expr]} ] [, ...] )
        [ DIM variable {, variable} ] ]
        [ STATIC variable {, variable} ] ]
        [ SHARED variable {, variable} ] ]
```

```

[ RETURN expr {, expr} ]
[ EXIT FUNCTION ]
  { instrucciones }
END FUNCTION

```

Las funciones en wxBasic brindan dos formas de devolver los valores. Al estilo Basic:

```

Function add( a, b )
add = a + b
End Function

```

O al estilo C:

```

Function add( a, b )
Return a + b
End Function

```

Al igual que C, se puede ignorar el Return. Entonces la función se comportará como una rutina.

Otro aspecto interesante que dota de más flexibilidad al lenguaje es que las funciones al igual que las expresiones pueden devolver más de un valor:

```

Function suma( a, b )
Return a, b, a + b
End Function

```

Para recoger los tres valores que devuelve tendremos que emplear tres variables:

```

varA,varB,sumaVar=suma(5,8)

```

Dentro de los parámetros se pueden incluir valores opcionales. Si estos valores no son incluidos en la llamada a la función, se les asigna los valores por defecto:

```

FUNCTION ConOpcionales( a, b=10, c="cadena por defecto" )
  ssMensaje("a=" & a)
  ssMensaje("b=" & b)
  ssMensaje("c=" & c)
END FUNCTION

```

Se pueden emplear variable estáticas (*static*) en la función. Estas son variables que retienen su valor aún después de que el compilador salga de la función donde fue creada. (vea STATIC)

Algunos o todos los valores que devuelve una función pueden ser ignorados. Si una función devuelve más valores que los requeridos, los valores extras son descartados. Por otra parte si la función devuelve menos valores que los esperados, las variables extras tomarán valor Nothing:

```

FUNCTION returnThreeValues()

```

```

        RETURN 1, 2, 3
    END FUNCTION

' ignora todos los valores
returnThreeValues()

' ignora el último valor
a, b = returnThreeValues

' Nothing es asignado a d
a, b, c, d = returnThreeValues()

```

wxBasic pasa los valores mediante los parámetros a las funciones por valor (ByVal) de modo que siempre se pasa una copia del valor de las variables, esto permite realizar la siguiente operación que provoca un intercambio de valores:

```
a, b = b, a
```

Clases

wxBasic permite un trabajo simple con clases que pueden contener campos, procedimientos, funciones, constructor, destructor, etc. Se debe tener presente que para los proyectos de HAEduc las clases se deben situar en el libro, mientras que si estamos en modo script (ver apartado *El Runtime*) las clases deben ser situadas al inicio del código y evitando que se declaren procedimientos antes de estas.

Veamos los ejemplos:

```

' creando una clase punto
CLASS Punto
    DIM x, y
END CLASS

' creando un punto
p = Punto()
p.x = 10
p.y = 20

```

En el ejemplo anterior las clases se emplean al estilo de las estructuras en C.

Los objetos son automáticamente destruidos cuando no hacen referencia a ningún objeto. Si se desea crear un objeto permanentemente se debe crear mediante la palabra reservada New:

```

' creando una clase punto
CLASS Punto
    DIM x, y
END CLASS

```

```
' crea un punto temporal y se le asigna algún valor
p1 = Punto ()
p1.x, p1.y = 10, 20
```

```
' crea un punto permanente y se le asigna algún valor
p2 = NEW Punto ()
p2.x, p2.y = 40, 80
```

```
' destruye el punto temporal
p1 = NOTHING
```

```
' Esto no destruye el punto permanente
p2 = NOTHING
```

Para destruir un objeto permanente emplee el símbolo: ~

```
' destruir el punto
~ p2
```

Es posible definir un inicializador para una clase incluyendo el método New en esta:

```
' crear una clase punto
CLASS Punto
  DIM x, y

  SUB NEW ( useX, useY )
    x, y = useX, useY
  END SUB
END CLASS
```

```
' Crear un punto
p1 = Punto( 10, 20 )
```

Cuando un objeto es destruido, la rutina FINALIZE de este, es llamada:

```
' crear una clase punto
CLASS Punto
  DIM x, y

  SUB NEW ( useX, useY )
    x, y = useX, useY
    ssMensaje("Punto creado {" & x & ", " & y & "}") )
  END SUB

  SUB FINALIZE()
    ssMensaje("Destruído el punto {" & x & ", " & y & "}") )
  END SUB
END CLASS
```

Las clases de wxBasic permiten un trabajo básico de herencia y la creación de clases abstractas:

```
class persona
```

```

    dim nombre as string
    dim edad as integer
end class

//clase alumno que hereda de la clase persona
class alumno inherits persona
    dim grado as integer

    sub new(nombre)
        me.nombre=nombre
    end sub

end class

un_alumno= alumno("Pedrito")
un_alumno.edad=14

```

En este ejemplo también está permitido crear un objeto del tipo persona:

```

una_persona=persona()
una_persona.nombre="Juan"

```

Si se desea crear clases abstractas, que no puedan ser instanciadas directamente sino que el programador sólo pueda emplearla para heredar otras clases entonces se emplea la palabra reservada `abstract`. de esta forma la clase persona quedaría:

```

abstract class persona
    dim nombre as string
    dim edad as integer
end class

```

De esta forma la siguiente línea generará un error:

```

una_persona=persona()

```

Clases Internas de Sora Script

Clase ssRect

Representa un área rectangular a partir de un punto (borde superior izquierdo) y el ancho y alto.

ssRect::ssRect

ssRect(x as **integer**, y as **integer**, alto as **integer**, ancho as **integer**) as **ssRect**

ssRect::obtancho

obtancho() as **integer**

Devuelve el ancho de un ssRect.

ssRect::obtalto**obtalto()** as **integer**

Devuelve el alto de un ssRect.

ssRect::estancho**estancho(w** as **integer)**

Establece el ancho de un ssRect.

ssRect::estalto**estalto(h** as **integer)**

Establece el alto de un ssRect.

ssRect::~~ssRect**~ssRect()**

Destructor de la clase

Clase *ssPunto*

Clase que representa un punto en el plano.

ssPunto::ssPunto**ssPunto(x** as **integer**, **y** as **integer**) as **ssPunto**

Donde x y y son las coordenadas del punto.

ssPunto::obt**obt()** as **integer**, as **integer**

Devuelve los valores de las coordenadas x y.

Ejemplo:

```
dim p = sspunto(10,11)
x,y=p.obt()
```

ssPunto::est**est(x** as **integer**, **y** as **integer**)

Establece los valores de las coordenadas x y.

ssPunto::obtx**obtx()** as **integer**

Devuelve el valor de la coordenada x del punto.

ssPunto::obty**obty()** as **integer**

Devuelve el valor de la coordenada y del punto.

ssPunto::~~ssPunto**~ssPunto()**

Destructor de la clase.

Clase *ssColor*

Clase para la gestión de colores. Contiene los componentes rojo, verde y azul de un color.

ssColor::ssColor

ssColor(r as **integer**, v as **integer**, a as **integer**) as **ssColor**

Donde *r,v,a* son las las componentes rojo, verde y azul que caracterizan a este color y sus valores están en el rango de 0 a 255.

ssColor::azul

azul() as **integer**

Devuelve el valor del componente azul de un ssColor.

ssColor::verde

verde() as **integer**

Devuelve el valor del componente verde de un ssColor.

ssColor::rojo

rojo() as **integer**

Devuelve el valor del componente rojo de un ssColor.

ssColor::ok

ok() as **integer**

Devuelve 1 si el ssColor es válido a partir del valor de sus componentes en el rango de 0 a 255.

ssColor::est

est(r as **integer**, v as **integer**, a as **integer**)

Establece los valores de los componentes de un ssColor.

ssColor::obt

obt() as **integer**, as **integer**, as **integer**

Devuelve el valor de los componentes rojo, verde y azul de un ssColor.

Ejemplo:

```
c=sscolor(100,255,150)
```

```
r,v,a=c.obt()
```

Clase ssCC

Clase para la gestión de cadenas de caracteres.

ssCC::ssCC

ssCC() as **ssCC**

Ejemplo:

```
dim cadena = ssCC()
```

ssCC::string

string() as **string**

Devuelve el contenido de un ssCC como el tipo de dato string.

Ejemplo:

```
dim cadena = ssCC()
```

```
cadena.append("texto")
```

dim str as string
str=cadena.string()

ssCC::reservar

reservar(n as integer)

Reserva previamente n bytes de memoria en el objeto.

ssCC::agregar

agregar(texto as string)

Agrega un texto al final del contenido de un ssCC.

ssCC::despuesprimer

despuesprimer(caracter as **string**) as **string**

Devuelve el texto que existe a continuación de la primera aparición de un *caracter* en un ssCC.

Nota: Solo se tomará en cuenta el primer *caracter* del string enviado como parámetro en este método.

ssCC::despuesultimo

despuesultimo(caracter as **string**) as **string**

Devuelve el texto que existe a continuación de la última aparición de un *caracter* en un ssCC.

Nota: Solo se tomará en cuenta el primer *caracter* del string enviado como parámetro en este método.

ssCC::antesprimero

antesprimero(caracter as **string**) as **string**

Devuelve el texto que existe antes de la primera aparición de un *caracter* en un ssCC.

Nota: Solo se tomará en cuenta el primer *caracter* del string enviado como parámetro en este método.

ssCC::antesultimo

antesultimo(caracter as **string**) as **string**

Devuelve el texto que existe antes de la última aparición de un *caracter* en un ssCC.

Nota: Solo se tomará en cuenta el primer *caracter* del string enviado como parámetro en este método.

ssCC::vaciar

vaciar()

Vacia el contenido de un ssCC.

ssCC::cmp

cmp(str as **ssCC**) as **integer**

Devuelve 1 si la cadena es mayor que el argumento str, cero si son iguales y -1 si es menor.

ssCC::cmpnocase

cmpnocase(str as **ssCC**) as **integer**

Esta comparación distingue entre mayúsculas y minúsculas.

Devuelve 1 si la cadena es mayor que el argumento str, cero si son iguales y -1 si es menor.

ssCC::contiene

contiene(str as **ssCC**) as **integer**

Devuelve 1 si el objeto ssCC contiene en cualquier parte de la cadena el argumento str.

ssCC::buscar

buscar(caracter as **string**, desdeelfinal as **integer**) as **integer**

Busca un caracter en la cadena y devuelve su posición. De no encontrarlo devuelve el valor ssNO_EXISTE.

Nota: Solo se tomará en cuenta el primer *caracter* del string enviado como primer parámetro en este método.

ssCC::freq

freq(caracter as **string**) as **integer**

Devuelve el número de veces que aparece un caracter en la cadena.

Nota: Solo se tomará en cuenta el primer *caracter* del string enviado como parámetro en este método.

ssCC::obtcaracter

obtcaracter(n as **integer**) as **string**

Devuelve el caracter que se encuentra en la posición n de la cadena.

ssCC::obtstring

obtstring() as **string**

Devuelve el contenido de la cadena.

ssCC::vacía

vacía() as **integer**

Devuelve 1 si la cadena esta vacía.

ssCC::esnumero

esnumero() as **integer**

Devuelve 1 si la cadena representa un valor numérico.

ssCC::espalabra

espalabra() as **integer**

Devuelve 1 si la cadena es una palabra.

ssCC::ultimo

ultimo() as **string**

Devuelve el último caracter de la cadena.

ssCC::izquierda

izquierda(n as **integer**) as **string**

Devuelve los n primeros caracteres de la cadena.

ssCC::longitud

longitud() as **integer**

Devuelve la longitud de la cadena.

ssCC::minusculas
minusculas() as **string**

Devuelve la cadena convertida a minúsculas.

ssCC::convertirminusculas
convertirminusculas()

Transforma el contenido del ssCC convirtiéndolo en minúsculas.

ssCC::convertirmayusculas
convertirmayusculas()

Transforma el contenido del ssCC convirtiéndolo en mayúsculas.

ssCC::subcadena

subcadena(pos as **integer**, ncaracteres as **integer**) as **string**

Devuelve una subcadena a partir del contenido del ssCC, que comienza en *pos* y tiene *ncaracteres* de longitud.

ssCC::pad

pad(n as **integer**, car as **string**=" ", fromRight as **integer** = true)

Agrega *n* caracteres *car* (por defecto = " ") en un extremo (derecho por defecto) de la cadena contenida en el ssCC.

ssCC::agregarinicio

agregarinicio(texto as **string**)

Agrega un texto al inicio de la cadena contenida en un ssCC.

ssCC::remover

remover(n as **integer**)

Elimina la porción de la cadena contenida en un ssCC desde el caracter *n* hasta el final.

ssCC::removerultimo

removerultimo()

Elimina el último caracter de la cadena.

ssCC::reemplazar

reemplazar(str1 as **string**, str2 as **string**, todos as **integer** = true) as **integer**

Reemplaza la cadena *str1* por *str2* en el contenido del ssCC. Si el parámetro *todos* es verdadero (por defecto lo es) se reemplazan todas las ocurrencias, de otro modo sólo será reemplazada la primera ocurrencia. Devuelve el número de reemplazos hechos.

ssCC::derecha

derecha(n as **integer**) as **string**

Devuelve los *n* caracteres existentes al final de la cadena.

ssCC::estcaracter

estcaracter(n as **integer**, char as **string**)

Establece el caracter de la posición *n* en la cadena. Solo se empleará el primer caracter del segundo parámetro.

ssCC::optimizar

optimizar()

Minimiza la memoria del ssCC. Se emplea para ajustar la memoria de la cadena si se le reservó anteriormente más de la realmente necesaria mediante el método *reservar*.

ssCC::comienzacon

comienzacon(inicio as **string**, resto as **ssCC**=NULL) as **integer**

Verifica que la cadena contenida en un ssCC comience con el texto del parámetro *inicio*, de ser así le establece al parámetro resto (si no es NULL) el resto de la cadena sin el prefijo contenido en *inicio*. Entonces devuelve 1, sino devuelve 0.

ssCC::finalizacon

finalizacon(final as **string**, resto as **ssCC**=NULL) as **integer**

Verifica que la cadena contenida en un ssCC finaliza con el texto del parámetro *final*, de ser así le establece al parámetro resto (si no es NULL) el resto de la cadena sin el sufijo contenido en *final*. Entonces devuelve 1, sino devuelve 0.

ssCC::anumber

anumber() as **number**

Devuelve el contenido del ssCC a number.

ssCC::trim

trim(fromRight as **integer** = true)

Elimina los espacios vacíos (espacios, saltos de línea, tabs, etc) en el extremo izquierdo o en el derecho (derecho por defecto) de la cadena contenida en el ssCC.

ssCC::truncar

truncar(pos as **integer**)

Elimina los caracteres a partir de la posición *pos* en la cadena del ssCC.

ssCC::mayusculas

mayusculas() as **string**

Devuelve el contenido del ssCC en mayúsculas.

Clase ssDim

Clase para la gestión de dimensiones. Muy empleada en la construcción de objetos.

ssDim::ssDim

ssDim(ancho as **integer**, alto as **integer**) as **ssDim**

ssDim::obtancho

obtancho() as **integer**

Devuelve el ancho de un `ssDim`.

ssDim::obtalto

obtalto() as **integer**

Devuelve el alto de un `ssDim`.

ssDim::est

est(ancho as integer, alto as integer)

Establece el ancho y alto de un `ssDim`.

ssDim::estancho

estancho(w as integer)

Establece el ancho de un `ssDim`.

ssDim::estalto

estalto(h as integer)

Establece el alto de un `ssDim`.

ssDim::~~ssDim

~ssDim()

Destructor de la clase.

Clase *ssFuente*

Clase para el manejo de fuentes.

ssFuente::ssFuente

ssFuente(dimension as **integer**, familia as **integer**, estilo as **integer**, grosor as **integer**, subrayado as **integer**= false, nombrefuente as **string**= "") as **ssFuente**

Donde:

dimensión es el tamaño de la fuente.

familia: Las familias de fuentes constituyen una forma de referirse a fuentes sin especificar el nombre. Ellas son:

<code>ssFONTFAMILY_DEFAULT</code>	(predeterminada)
<code>ssFONTFAMILY_DECORATIVE</code>	
<code>ssFONTFAMILY_ROMAN</code>	
<code>ssFONTFAMILY_SCRIPT</code>	
<code>ssFONTFAMILY_SWISS</code>	
<code>ssFONTFAMILY_MODERN</code>	
<code>ssFONTFAMILY_TELETYPE</code>	

El estilo puede tomar los valores:

`ssFONTSTYLE_NORMAL`
`ssFONTSTYLE_SLANT`
`ssFONTSTYLE_ITALIC`

El grosor puede tomar los valores:

`ssFONTWEIGHT_NORMAL`
`ssFONTWEIGHT_LIGHT`

ssFONTWEIGHT_BOLD

El *nombrefuente* permite especificar el nombre de la fuente a emplear. De no existir o dejarse vacío este parámetro, se empleará la fuente por defecto basada en la familia seleccionada.

ssFuente::obtnombrefuente
obtnombrefuente() as **string**

Devuelve el nombre de la fuente o una cadena vacía si no tiene información de este parámetro.

ssFuente::obtfamilia
obtfamilia() as **string**

Devuelve el nombre de la familia.

ssFuente::obtdimension
obtdimension() as **integer**

Devuelve el tamaño de la fuente.

ssFuente::obtestilo
obtestilo() as **integer**

Devuelve el estilo de la fuente.

ssFuente::obtsubrayado
obtsubrayado() as **integer**

Devuelve si la fuente tiene estilo subrayado.

ssFuente::obtgrosor
obtgrosor() as **integer**

Devuelve el grosor de la fuente.

ssFuente::estnombrefuente
estnombrefuente(nombrefuente as **string**)

Establece el nombre de la fuente.

ssFuente::estfamilia
estfamilia(familia as **integer**)

Establece la familia de la fuente.

La familia puede tomar los valores:

ssFONTFAMILY_DEFAULT (predeterminada)

ssFONTFAMILY_DECORATIVE

ssFONTFAMILY_ROMAN

ssFONTFAMILY_SCRIPT

ssFONTFAMILY_SWISS

ssFONTFAMILY_MODERN

ssFONTFAMILY_TELETYPE

ssFuente::estdimension
estdimension(dimension as **integer**)

Establece el tamaño de la fuente.

ssFuente::estestilo**estestilo**(estilo as **integer**)

Establece el estilo de la fuente.

El estilo puede tomar los valores:

ssFONTSTYLE_NORMAL

ssFONTSTYLE_SLANT

ssFONTSTYLE_ITALIC

ssFuente::estsubrayado**estsubrayado**(subrayado as **integer**)

Establece el subrayado de la fuente.

ssFuente::estgrosor**estgrosor**(grosor as **integer**)

Establece el grosor de la fuente.

El grosor puede tomar los valores:

ssFONTWEIGHT_NORMAL

ssFONTWEIGHT_LIGHT

ssFONTWEIGHT_BOLD

Sora Script contiene por defecto las siguientes fuentes que pueden ser empleadas directamente sin necesidad de crearlas:

ssNORMAL_FONT

ssSMALL_FONT

ssITALIC_FONT

ssSWISS_FONT

Ejemplo:

mifuentes=ssITALIC_FONT

Clase ssNullImagenBase

ssNullImagenBase::ssNullImagenBasessNullImagenBase() as **ssImagenBase**Hereda de **ssImagenBase**Constructor de la clase **ssNullImagenBase**, que representa un objeto **ssImagenBase** vacío.

Ejemplo:

dim imagenvacia= new ssNullImagenBase()

Clase ssImagenBase

ssImagenBase::ssImagenBasessImagenBase(ima as **ssImagenBase**) as **ssImagenBase**Constructor de la clase **ssImagenBase**. Para crear un objeto **ssImagenBase** vacío se puede pasar el parámetro **ima** como un **ssNullImagenBase**.

Ejemplo:

```
unaimagen =ssImagenBase(ssNullImagenBase())
```

ssImagenBase::cargarfichero

cargarfichero(file as **string**, tipo as **integer**=ssBITMAP_TYPE_ANY) as **integer**
Carga un fichero de imagen. El parámetro tipo define el formato de lectura del fichero y puede tener los siguientes valores:

```
ssBITMAP_TYPE_BMP
ssBITMAP_TYPE_ICO
ssBITMAP_TYPE_CUR
ssBITMAP_TYPE_XBM
ssBITMAP_TYPE_XPM
ssBITMAP_TYPE_TIF
ssBITMAP_TYPE_GIF
ssBITMAP_TYPE_PNG
ssBITMAP_TYPE_JPEG
ssBITMAP_TYPE_PNM
ssBITMAP_TYPE_PCX
ssBITMAP_TYPE_PICT
ssBITMAP_TYPE_ICON
ssBITMAP_TYPE_MACCURSOR
ssBITMAP_TYPE_ANY
```

ssImagenBase::xpmaimagen

xpmaimagen(datos as **table**)

Carga una tabla de valores que contiene la información de una imagen similar al formato xpm.

Ejemplo:

```
cursor = {
"11 15 37 1",
"      c #EFEFEF",
".      c #B9BABB",
"+      c #8C8D90",
"@      c #F4F4F6",
"#      c #F1F2F4",
"$      c #E8E9EF",
"%      c #DADCE1",
"&      c #EFF0F2",
"*      c #B7B9BC",
"=      c #E6E7ED",
"-      c #D8DADF",
";      c #EDEEF1",
">      c #F5F6F7",
",      c #E3E4EA",
"      c #D5D7DC",
")      c #EBECEf",
"!      c #E1E2E8",
"~      c #D3D5DA",
"{      c #E9EAED",
"}      c #DEDFE5",
"^      c #D1D2D7",
"/      c #E7E8EC",
"(      c #DCDDE3",
"      c #CED0D5",
":      c #E5E6EA",
"<      c #DADBE1",
"[      c #CCDD2",
"}      c #E3E4E8",
"|      c #D7D8DE",
"1      c #CACBD0",
"2      c #BCBDBE",
"3      c #D5D6DC",
"4      c #D2D3D9",
```

```
"5      c #D0D1D7",
"6      c #CECFD5",
"7      c #D0D0D1",
"8      c #9A9B9D",
".+++++++",
".+@@@@@+.",
"+#$$$$$%+",
"+&***==*-+",
"+;*>,,,*>'+",
"+)!!!!!!~+",
"+{}}]}}]{}^+",
"+/(((((((+_+",
"+:<<<<<<<[+",
"+}|||||1+",
"2+3333333+2",
" 2+444444+2 ",
" 2+555+2 ",
" 2+6+2 ",
" 787  "}
```

```
//creando el objeto sslmagenBase
unaimagen =sslmagenBase(ssNulllmagenBase())
//cargando los datos en el objeto sslmagenBase
unaimagen.xpmaimagen(cursor)
```

sslmagenBase::~sslmagenBase

~sslmagenBase()

Destructor de los objetos de esta clase.

Ejemplo:

```
unaimagen =sslmagenBase(ssNulllmagenBase())
~unaimagen
```

Nota importante:

El empleo de la clase sslmagenBase es de gran utilidad combinándolo con el objeto Imagen y botón del proyector. Dichos objetos contienen métodos (por ejemplo EstlmagenIB) que le permiten cargar el contenido de un sslmagenBase. Esto último unido con el método *xpmaimagen* permite tener imágenes empotradas en las aplicaciones en forma de script, eliminando así las dependencias externas a estos recursos de imagen y aumentando la velocidad de carga.

Clase ssSonido

Clase para la reproducción de sonidos de diversos formatos (mp3, wav, etc) a nivel de libro. Esta clase es empleada particularmente para reproducir y control de sonidos que puedan ser escuchados durante la navegación entre páginas.

ssSonido::ssSonido

ssSonido(nombre as **string**, fichero as **string**) as **ssSonido**

Constructor de la clase ssSonido. Para construir un objeto ssSonido puede pasarse el parámetro fichero con el valor "" y luego emplear el método cargar. El parámetro nombre será el identificador empleado para la captura de los eventos como mismo sucede con los objetos del proyector.

ssSonido::reproducir

reproducir()

Inicia la reproducción de un ssSonido.

ssSonido::pausar**pausar()**

Hace una pausa en la reproducción de un ssSonido.

ssSonido::detener**detener()**

Detiene la reproducción de un ssSonido.

ssSonido::obtvolumen**obtvolumen()** as **Number**

Devuelve el volumen de un ssSonido.

ssSonido::obtestado**obtestado()** as **integer**

Devuelve el estado actual de un ssSonido. Los valores devueltos pueden ser:

```
SS_DETENIDO
SS_PAUSA
SS_REPRODUCIENDO
```

ssSonido::cargar**cargar**(fichero as **string**) as **integer**

Devuelve 1 si el ssSonido pudo cargar un fichero.

ssSonido::posicionar**posicionar**(pos as **integer**) as **integer**

Devuelve 1 si el ssSonido pudo posicionar su reproducción en la posición pos.

ssSonido::estvolumen**estvolumen**(vol as **number**)

Establece el volumen de ssSonido. Los valores admitidos van desde 0 hasta 1.

ssSonido::obtposicion**obtposicion()** as **integer**

Devuelve la posición de reproducción de un ssSonido.

ssSonido::obtduracion**obtduracion()** as **integer**

Devuelve la duración de un fichero cargado en un ssSonido.

Trabajo con ficheros

wxBasic permite el tratamiento de ficheros mediante acceso aleatorio, binario y secuencial.

OPEN

OPEN fichero FOR modo AS #canal

Abre un fichero para lectura, escritura y escritura manteniendo su contenido. Esta sintaxis es algo incómoda y antigua, preferiblemente emplee FOpen().

Los modos de abrir un fichero son:

- Input Leer desde un fichero
- Output Escribir en el fichero destruyendo el contenido anterior
- Append Escribir en el fichero manteniendo el contenido anterior

El canal es el número que identifica el fichero. Se puede obtener un canal libre llamando a la función FreeFile().

Cierre un fichero mediante Close # o Fclose(). Es posible además cerrar todos los ficheros abiertos mediante Close.

```
' copiar "source.txt" a "copy.txt"
OPEN "source.txt" FOR INPUT AS #1
OPEN "copy.txt" FOR OUTPUT AS #2

' leer el primer fichero e ir copiándolo al segundo
WHILE NOT EOF( 1 )
    LINE INPUT #1, text
    PRINT #2, text
WEND

' cerrar los ficheros
CLOSE #1
CLOSE #2
```

SHARED

SHARED variable {, variable }

Declara una variable en una función o Sub haciendo referencia a una variable global de igual nombre, permitiendo acceder a la global mediante dicha variable local.

```
DIM MiVariableGlobal

FUNCTION MiFuncion()
    SHARED MiVariableGlobal
    ssMensaje("el valor de la variable global es " & MiVariableGlobal)
END FUNCTION
```

STATIC

STATIC variable {, variable }

Al declarar una variable local de una Función o Sub como STATIC su valor será mantenido por el compilador incluso después de haber salido de dicha función o sub. El valor inicial de una variable STATIC es Nothing.

```

FUNCTION accum( n )
    ' se declara result como una variable STATIC
    STATIC result

    ' ¿primera llamada a accum?
    IF result = NOTHING THEN
        ' initialize the result
        result = n
    ELSE
        ' acumular valores en result
        result = result + n
    END IF

    ' devolver el resultado acumulado
    RETURN result
END FUNCTION

```

PRINT

PRINT # expr { [expr] [,] [;] }

Escribe una cadena o el valor de una expresión en un fichero a través del handler del fichero previamente abierto:

```

' abrir un fichero para escritura
OPEN "output.txt" FOR OUTPUT AS #1

' escribe texto en el fichero
PRINT #1, "Esto es escrito en el fichero"
PRINT #1, "y también esto"

' cerrar el fichero
CLOSE #1

```

Otras funciones para el trabajo con ficheros y directorios

::sconcatenarficheros

sconcatenarficheros(fichero1 as **string**, fichero2 as **string**, resultado as **string**) as **integer**
Concatena dos ficheros (fichero1 y fichero2) en un uno (resultado) y devuelve 0 o 1 según el resultado de la operación.

::sscopiafichero

sscopiafichero(fichero1 as **string**, fichero2 as **string** [, reescribir=1 as **integer**]) as **integer**

Hace una copia de fichero1 en fichero2 con la opción de reescribir activada por defecto. Devuelve 0 o 1 según el resultado de la operación.

::ssexistedirectorio

ssexistedirectorio(directorio as **string**) as **integer**
Verifica la existencia de un directorio.

::ssolodirectorio

ssolodirectorio(directorio as **string**) as **string**
Dada la dirección de un fichero, devuelve sólo el directorio, eliminando el nombre del fichero de la cadena pasada por el parámetro *directorio*.

::ssdirectorioabsoluto

ssdirectorioabsoluto(directorio as **string**) as **integer**
Devuelve 1 si el parámetro pasado constituye un directorio absoluto.

::ssdirectorioso

ssdirectorioso() as **string**
Devuelve el directorio del sistema operativo.

::ssobthome

ssobthome() as **string**
Devuelve el directorio Home.

::ssestdirectoriotrabajo

ssestdirectoriotrabajo(directorio as **string**)
Establece el directorio de trabajo del sistema.

::ssobtdirectoriotrabajo

ssobtdirectoriotrabajo() as **string**
Devuelve el directorio actual de trabajo del sistema.

Otras Funciones de Sora Script

A continuación se describen las funciones que se adicionaron al compilador original de wxBasic para controlar los elementos que componen al proyector y fundamentalmente para el trabajo con los objetos, las páginas, el libro, etc.

Funciones de navegación entre páginas

::ssnavegar

ssnavegar(*pag* as **integer**) as **integer**
Permite navegar a una página conociendo su posición (*pag*) en la lista de páginas. Si la página existe devuelve 1, sino 0.

::ssnavegarapagina

ssnavegarapagina (*pag* as **string**) as **integer**
Permite navegar a una página conociendo nombre (*pag*). Si la página existe devuelve 1, sino 0.

::ssnavegarsiguiente

ssnavegarsiguiente () as **integer**
Permite navegar a la siguiente página. Si la página existe devuelve 1, sino 0.

::ssnavegaranterior

ssnavegaranterior () as **integer**

Permite navegar a la página anterior. Si la página existe devuelve 1, sino 0.

::ssexistepagina

ssexistepagina (*pag* as **integer**) as **integer**

Verifica la existencia de una página conociendo su posición (*pag*). Si la página existe devuelve 1, sino 0.

::ssobtposicionpagina

ssobtposicionpagina (*pag* as **string**) as **integer**

Devuelve la posición de una página conociendo su nombre (*pag*), si no existe devuelve -1.

::ssnavegaambitopagina

ssnavegaambitopagina (*pag* as **string**)

Esta función se emplea fundamentalmente en scripts que son ejecutados en el runtime para simular un cambio de ámbito de página. Para más información ver el apartado *El Runtime*.

::sspaginaestfondo

sspaginaestfondo(*fichero* as **string**) as **integer**

Establece la imagen de fondo del proyector. Devuelve 0 o 1 según el resultado de la operación.

Otras funciones

::ssestmodovideo

ssestmodovideo(*ancho* as **integer**, *alto* as **integer**, *pcolores* as **integer**, *refresh* as **integer**) as **integer**

Permite cambiar la resolución de la pantalla. Este método puede fallar para algunas distribuciones de Linux. Devolverá 1 si logra hacer el cambio de resolución, sino devuelve 0.

::sspantallacompleta

sspantallacompleta(*[pcompleta=true* as **integer]) as **integer****

Redimensiona la aplicación al tamaño de la pantalla, devuelve 1 si lo logra, de lo contrario devuelve 0.

::sspunteroobtposicion

sspunteroobtposicion() as **integer**, as **integer**

Devuelve la posición del puntero del mouse en la aplicación.

::ssredibujar

ssredibujar()

Provoca el redibujado de toda el área del proyector

::ssredibujarrectangulo

ssredibujarrectangulo(*x* as **integer**, *y* as **integer**, *ancho* as **integer**, *alto* as **integer**)

Provoca el redibujado del área del proyector comprendida en el rectángulo que tiene origen en *x,y* con un ancho y alto específico.

::ssmostrarmarco

ssmostrarmarco(*mostrar=true* as **integer**)

Muestra u oculta un fondo tras la aplicación.

::ssmarcoestcolor

ssmarcoestcolor(*color* as **ssColor**)

Establece el color del fondo tras la aplicación que por defecto es negro.

::ssobtvambiente

ssobtvambiente(*nombre* as **string**) as **string**

Devuelve el valor de una variable de ambiente.

Ejemplo:

```
//este ejemplo devuelve la ruta del directorio Home en linux
```

```
ssmensaje(ssobtvambiente("HOME"))
```

::ssetvambiente

ssetvambiente(nombre as **string**, valor as **variant**)

Establece el valor de una variable de ambiente. Si no existe la crea.

::sseliminarvambiente

sseliminarvambiente(nombre as **string**)

Elimina una variable de ambiente.

::ssfechamodificacionfichero

ssfechamodificacionfichero(fichero as **string**) as **string**

Devuelve la fecha de modificacion de un fichero.

::ssfichero deruta

ssfichero deruta(ruta as **string**) as **string**

Devuelve el nombre de un fichero extraido de una ruta (no necesariamente debe existir dicha ruta).

Ejemplo:

```
fic=ssfichero deruta("/home/usuario/leeme.txt")
```

//fic toma el valor "leeme.txt"

::ssbuscarprimerfichero

ssbuscarprimerfichero(ruta as **string**) as **string**

Devuelve el primer fichero contenido en una ruta.

::ssbuscarproximofichero

ssbuscarproximofichero(ruta as **string**) as **string**

Devuelve el próximo fichero contenido en una ruta.

Ejemplo:

```
//este ejemplo permite recorrer cada uno de los ficheros contenidos en /usr/local
```

```
ss =ssbuscarprimerfichero("/usr/local/*")
```

```
tt = ss
```

```
while len(ss)>0
```

```
    ss=ssbuscarproximofichero()
```

```
    tt = tt & ss & "\n"
```

```
end while
```

```
ssmensaje(tt)
```

::ssreproducirwav

ssreproducirwav(fichero as string[,sinc as integer=SS_ASINCRONICO])

Reproduce un fichero de sonido de formato WAV de forma sincrónica o asincrónica

El parámetro *sinc* puede tomar los valores:

SS_SINCRONICO reproducción sincrónica

SS_ASINCRONICO reproducción asincrónica

::ssejecutar

ssejecutar(comandos as **string**[, sinc as **integer=0**]) as **integer**

Ejecuta comandos.

Los valores que puede tomar el parámetro sinc son:

0 Ejecución asincrónica

1 Ejecución sincrónica

2 Ejecución visible

3 Ejecución no deshabilitada

::ssresultadoejecucion

ssresultadoejecucion(comandos as **string**) as **string**

Devuelve la salida de errores que genera un ejecutable al cerrarse. Este método puede ser empleado para obtener información del funcionamiento de un programa ejecutado al cerrarse.

::ssconsola

ssconsola(comandos as **string**) as **integer**
Ejecuta un comando de consola y devuelve el resultado.

::ssaplicacionestmascara

ssaplicacionestmascara(fichero as **string**, color as ssColor, tolerancia as **integer**) as **integer**
Permite establecerle una máscara a la aplicación a partir de un fichero de imagen. Los parámetros rojo, verde y azul son el color que se tornará transparente de la superficie de la aplicación. El parámetro *tolerancia* selecciona hasta que intensidad del color seleccionado se tornará transparente. Devuelve 0 o 1 si logra realizar la operación.

::ssaplicacionestitulo

ssaplicacionestitulo(titulo as **string**)
Establece el título de la ventana de la aplicación

::ssaplicacionestpuntero

ssaplicacionestpuntero(puntero as [**integer** | **string**])
Establece el puntero de la aplicación pasándole como parámetro el valor de un puntero predeterminado o la dirección de una imagen. Los punteros predeterminados son:

SS_PUNTERO_MANO	Representa una mano
SS_PUNTERO_EDICION	Puntero de edición de texto
SS_PUNTERO_PROHIBIDO	Símbolo de prohibición
SS_PUNTERO_INTERROGACION	Símbolo de interrogación
SS_PUNTERO_INTERROGACION	Flecha con dirección norte-sur y este-oeste
SS_PUNTERO_NS	Flecha con dirección norte-sur
SS_PUNTERO_NOSE	Flecha con dirección noroeste-sureste
SS_PUNTERO_EO	Flecha con dirección este-oeste
SS_PUNTERO_CRUZ	Símbolo en forma de cruz
SS_PUNTERO_ESPERA	Símbolo de espera
SS_PUNTERO_FLECHAESPERA	Símbolo de espera con flecha
SS_PUNTERO_FLECHA	Símbolo flecha (puntero normal)
SS_PUNTERO_IZQUIERDO	Símbolo flecha invertida
SS_PUNTERO_DERECHO	Símbolo flecha horizontal hacia la derecha

::ssaplicacionmostrar

ssaplicacionmostrar()
Muestra la ventana de la aplicación si está oculta.

::ssaplicacionocultar

ssaplicacionocultar()
Oculta la ventana de la aplicación si está visible

::ssaplicacionestilomascara

ssaplicacionestilomascara(ancho as **integer**, alto as **integer**)
Establece una máscara rectangular a la aplicación de dimensiones *ancho* y *alto*. Devuelve 0 o 1 según el resultado de la operación.

::ssaplicacioncentrar

ssaplicacioncentrar()
Posiciona la ventana de la aplicación en el centro de la pantalla.

::ssespera

ssespera(n as **integer**)
Provoca una espera en el flujo de la aplicación en *n* milisegundos

::ssaplicacionestposicion

ssaplicacionestposicion(x as **integer**, y as **integer**)
Establece la posición de la aplicación en la pantalla.

::ssaplicacionobtposicion

ssaplicacionobtposicion() as **integer**, as **integer**
Devuelve la posición de la aplicación en la pantalla

::ssaplicacionestdimensiones

ssaplicacionestdimensiones (largo as **integer**, alto as **integer**)
Establece las dimensiones de la aplicación en la pantalla.

::ssaplicacionsalir

ssaplicacionsalir()
Cierra la aplicación.

::ssversionso

ssversionso() as **integer**

Devuelve la versión del sistema operativo sobre el que se esta ejecutando la aplicación. Para Windows devuelve la constante SS_SOWINDOWS mientras que en Linux devuelve SS_SOGNULINUX. Existe además una constante del sistema nombrada SO que contiene uno de estos valores y permite realizar la siguiente comparacion:

```
if (SO = SS_SOWINDOWS) then
    //estamos en windows
else if(SO= SS_SOGNULINUX) then
    //estamos en Linux
end if
```

::sspunteroestposicion

sspunteroestposicion(x as integer, y as integer)
Sitúa el puntero del mouse en una posición.

::sslanzarnavegador

sslanzarnavegador(dirección as string) as **integer**

Lanza el navegador por defecto con una dirección. Devuelve 0 o 1 según el resultado de la operación.

::ssobtmovideo

ssobtmovideo() as **integer**, as **integer**

Devuelve la resolución actual de la pantalla.

Ejemplo:

Largo, ancho= ssobtmovideo()

::ssllamada

ssllamada(padre as **object**, procedimiento as **string** [...]) as **integer**

Intenta realizar una llamada a un procedimiento. Devuelve 1 si lo consigue, de lo contrario devuelve 0. El parámetro padre puede tomar el valor 0 (o nothing) si el procedimiento al que se desea llamar se encuentra en el ámbito global, sino debe especificarse el objeto al que le pertenece este procedimiento. Los demás parámetros son los parámetros que se empleará para realizar la llamada y no pueden ser un número mayor de 20.

Nota:

Este método es especialmente útil para automatizar llamadas y en la generación de eventos de clases de usuarios en las que intervengan o no objetos del proyector.

::cargarrecurso

cargarrecurso(as **string**)as **integer**

Agrega e interpreta un fichero de código en tiempo de ejecución incorporando el script que contiene al ámbito global. El código incluido tendrá acceso a los procedimientos, variables y clases del código ya interpretados. Para ejecutar un procedimiento que se encuentre en un fichero de código cargado emplee el método *ssllamada* estableciendo el primer parámetro a nothing y especificando el nombre del procedimiento y los parámetros (ver *ssllamada*).

::obtoobjeto

obtoobjeto(nombreobj as **string**) as **ssObjeto**

Devuelve el puntero hacia un objeto del proyector cuyo nombre sea nombreobj.

De existir más de un objeto con dicho nombre se tomará el primero según el orden de creación. Si no existe un objeto con dicho nombre el objeto no modifica su valor.

Nota: Este método es empleado para obtener una referencia a los objetos creados en modo de edición mediante el IDE de HAEduc. Es el único modo de acceder a estos objetos desde script.

Ejemplo:

```
//se supone que existe un objeto creado desde el ide de HAEduc con nombre "boton1"
dim bot=obtoobjeto("boton1")
if(bot) then
    bot.esttexto("Vertical")
    bot.rotar(90)
end if
```

Diálogos comunes

Los diálogos de Sora Script son clases internas que permiten el intercambio de información entre el usuario y las aplicaciones. Estos diálogos son:

```
ssDialogoColor
ssDialogoFichero
ssDialogoDirectorio
ssDialogoTexto
ssDialogoFuente
ssDialogoMensaje
ssMensaje
```

clase *ssDialogoColor*

Clase para la selección de colores por el usuario.

ssDialogoColor

ssDialogoColor::ssDialogoColor([color as **ssColor**]) as **ssDialogoColor**

Crea un cuadro de diálogo para la selección de colores. La apariencia del diálogo dependerá del sistema operativo (Windows o Linux).

ssDialogoColor::mostrar

mostrar() as **integer**

Muestra un diálogo de color previamente creado y devuelve los valores SS_ACEPTAR o SS_CANCELAR dependiendo del botón pulsado por el usuario al cerrar el diálogo.

ssDialogoColor::obtcolor

obtcolor() as **sscolor**

Devuelve el color seleccionado en la ventana de diálogo de color.

Ejemplo:

```
//color inicial negro
color=sscolor(0,0,0)
dialogo=ssDialogoColor(color)
if dialogo.mostrar()=SS_ACEPTAR then
    color=dialogo.obtcolor()
    ssMensaje("El color seleccionado fue: rojo>" & color.rojo() & " verde>" & color.verde() & " azul>" &
color.azul())
else
    ssmensaje("Accion cancelada")
end if
```

clase *ssDialogoFichero*

Clase para la selección de ficheros por el usuario.

ssDialogoFichero::ssDialogoFichero

ssDialogoFichero([título as **string**="Seleccione un fichero", directorio as **string**="", fichero as **string**="", filtro as **string**="*", estilo as **integer** = SS_ABRIR, pos as **ssPunto**=ssPosicionPorDefecto]) as **ssDialogoFichero**

Genera un cuadro de diálogo para seleccionar ficheros, empleado generalmente para las operaciones de abrir y guardar. Por defecto el estilo es de abrir.

Parámetros:

<i>título</i>	Título de la ventana de diálogo.
<i>directorio</i>	Directorio explorado por el diálogo al iniciarse.
<i>fichero</i>	Fichero seleccionado previamente en el diálogo.
<i>filtro</i>	Filtro de ficheros por nombre y extensión.

Ejemplo de filtro:

"All files (*.*)|*.*|JPEG files (*.jpg)|*.jpg"

<i>estilo</i>	Estilo del diálogo, puede tomar los valores:
SS_ABRIR	Para abrir un fichero.
SS_GUARDAR	Para guardar un fichero.
<i>pos</i>	Posición en la que se mostrará el diálogo.

ssDialogoFichero::obtdirectorio

obtdirectorio() as **string**

Devuelve el directorio por defecto del diálogo.

ssDialogoFichero::obtnombrefichero

obtnombrefichero() as **string**

Devuelve el fichero por defecto de diálogo.

ssDialogoFichero::obtmensaje

obtmensaje() as **string**

Devuelve el mensaje que será mostrado en el diálogo.

ssDialogoFichero::obtruta

obtruta() as **string**

Devuelve la ruta completa (directorio y nombre del fichero) del fichero seleccionado.

ssDialogoFichero::obtestilo

obtestilo() as **integer**

Devuelve el estilo de creación del diálogo.

ssDialogoFichero::obtfiltro

obtfiltro() as **string**

Devuelve el filtro del diálogo.

ssDialogoFichero::estdirectorio

estdirectorio(dir as **string**)

Establece el directorio por defecto del diálogo.

ssDialogoFichero::estnombrefichero

estnombrefichero(fichero as **string**)

Establece el fichero por defecto del diálogo.

ssDialogoFichero::estmensaje

estmensaje(mensaje as **string**)

Establece el mensaje que va a ser mostrado por el diálogo.

ssDialogoFichero::estruta

estruta(ruta as **string**)

Establece la ruta que será devuelta cuando el diálogo sea cerrado.

ssDialogoFichero::estestilo

estestilo(estilo as **integer**)

Establece el estilo que tendrá el diálogo al mostrarse.

ssDialogoFichero::estfiltro

estfiltro(filtro as **string**)

Establece el filtro que tendrá el diálogo para la selección de los ficheros.

ssDialogoFichero::mostrarmostrar() as **integer**

Muestra un diálogo de fichero previamente creado y devuelve los valores SS_ACEPTAR o SS_CANCELAR dependiendo del botón pulsado por el usuario al cerrar el diálogo.

clase ssDialogoDirectorio

Clase para la selección de directorios por el usuario.

ssDialogoDirectorio::ssDialogoDirectoriossDialogoDirectorio([título as **string**="", directorio as **string**="", pos as **sspunto** = ssPosicionPorDefecto]) as **ssDialogoDirectorio**

Muestra un diálogo para la selección de directorios.

Parámetros:

<i>título</i>	Título del diálogo.
<i>directorio</i>	Directorio por defecto en el que abrirá el diálogo.
<i>pos</i>	Posición donde aparecerá el diálogo.

ssDialogoDirectorio::obtdirectorioobtdirectorio() as **string**

Devuelve el directorio por defecto o el seleccionado por el usuario en el diálogo.

ssDialogoDirectorio::obtmensajeobtmensaje() as **string**

Devuelve el mensaje que será mostrado en el diálogo.

ssDialogoDirectorio::estmensajeestmensaje(mensaje as **string**)

Establece el mensaje que va a ser mostrado por el diálogo.

ssDialogoDirectorio::estdirectorioestdirectorio(dir as **string**)

Establece el directorio por defecto del diálogo.

ssDialogoDirectorio::mostrarmostrar() as **integer**

Muestra un diálogo de directorio previamente creado y devuelve los valores SS_ACEPTAR o SS_CANCELAR dependiendo del botón pulsado por el usuario al cerrar el diálogo.

clase ssDialogoTexto

Clase para la creación de cajas de diálogo de entrada de texto.

ssDialogoTexto::ssDialogoTextossDialogoTexto(contenido as **string** [, título as **string**= "Entre el texto", valorpordefecto as **string**="", pos as **sspunto**= ssPosicionPorDefecto]) as **ssDialogoTexto**

Genera un cuadro de diálogo que permite introducir textos a la aplicación.

Parámetros:

<i>contenido</i>	Texto de contenido.
<i>título</i>	Título de la ventana de diálogo.
<i>valorpordefecto</i>	Valor inicial por defecto del texto de entrada.
<i>pos</i>	Posición donde aparecerá la ventana.

ssDialogoTexto::obtvalorobtvalor() as **string**

Devuelve el valor introducido al cerrar la ventana de diálogo.

ssDialogoTexto::estvalorestvalor(valor as **string**)

Establece el valor por defecto del diálogo.

ssDialogoTexto::mostrarmostrar() as **integer**

Muestra un diálogo de entrada de texto previamente creado y devuelve los valores SS_ACEPTAR o SS_CANCELAR dependiendo del botón pulsado por el usuario al cerrar el diálogo.

clase ssDialogoFuente

Clase para la selección de fuentes por el usuario.

ssDialogoFuente::ssDialogoFuentessDialogoFuente([fuente as **ssfuente**]) as **ssDialogoFuente**

Crea un diálogo para la selección de fuentes.

ssDialogoFuente::obtfuenteobtfuente() as **ssfuente**

Devuelve la fuente seleccionada por el usuario.

ssDialogoFuente::mostrarmostrar() as **integer**

Muestra un diálogo de selección de fuente previamente creado y devuelve los valores SS_ACEPTAR o SS_CANCELAR dependiendo del botón pulsado por el usuario al cerrar el diálogo.

clase ssDialogoMensaje

Clase para mostrar cajas de mensajes.

ssDialogoMensaje::ssDialogoMensaje

ssDialogoMensaje(contenido as **string** [, título="Mensaje" as **string**, icono=SS_IINFORMACION as **integer**, botones=SS_BACEPTAR as **integer**]) as **ssDialogoMensaje**

Crea un diálogo de mensajes.

Parámetros:

<i>contenido</i>	Contenido del mensaje.
<i>título</i>	Título de la ventana de mensaje.
<i>icono</i>	Icono de la ventana, puede tener los valores:
SS_IEXCLAMACION	Muestra un icono de exclamación
SS_IERROR	Muestra un icono de error
SS_IPREGUNTA	Muestra un icono de interrogación
SS_IINFORMACION	Muestra un icono de Información
<i>botones</i>	Botones en la ventana, puede tomar los valores:
SS_BACEPTAR	Botón Aceptar
SS_BSINO	Dos botones, Sí y No
SS_BACEPTARCANCELAR	Dos botones, Aceptar y Cancelar
SS_BSINOCANCELAR	Tres botones, Sí, No y Cancelar

Nota: Una forma abreviada de generar mensajes se logra con el empleo del metodo ssMensaje.

::ssMensaje

ssMensaje(*contenido* as **string** [, título="Mensaje" as **string**, icono=SS_IINFORMACION as **integer**, botones=SS_BACEPTAR as **integer**]) as **integer**

Método para generar ventanas de mensajes similar a la clase ssDialogoMensaje. Genera un mensaje con un contenido, un título con valor por defecto "Mensaje", un icono de información por defecto y un botón Aceptar. Solo el primer parámetro es obligatorio. Los parámetros *icono* y *botones* pueden ser modificados con los siguientes valores que constituyen constantes de Sora Script:

Iconos:

SS_IEXCLAMACION	Muestra un icono de exclamación
SS_IERROR	Muestra un icono de error
SS_IPREGUNTA	Muestra un icono de interrogación
SS_IINFORMACION	Muestra un icono de Información

Combinación de Botones:

SS_BACEPTAR	Botón Aceptar
SS_BSINO	Dos botones, Sí y No
SS_BACEPTARCANCELAR	Dos botones, Aceptar y Cancelar
SS_BSINOCANCELAR	Tres botones, Sí, No y Cancelar

Dependiendo de los botones empleados para la construcción del diálogo y del botón pulsado, este devolverá las constantes:

Botones:

SS_BACEPTAR	Se pulsó el botón Aceptar
SS_CANCELAR	Se pulsó el botón Cancelar
SS_SI	Se pulsó el botón Sí
SS_NO	Se pulsó el botón No

Si se cierra en diálogo por la pestaña cerrar devolverá SS_CANCELAR.

Ejemplo

```

resultado=ssMensaje("De acuerdo?","Pregunta",SS_IPREGUNTA, SS_BSINO)
if resultado=SS_SI then
    ssMensaje("Estas de acuerdo")
else
    ssMensaje("No estas de acuerdo")
end if

```

Los Objetos básicos del proyector

El proyector de HaEduc brinda 11 objetos diferentes para el diseño de las aplicaciones:

1. Imagen
2. Botón
3. Etiqueta
4. Caja de texto
5. Html
6. Video
7. Polígono
8. Lista
9. Reloj
10. Tabla
11. Texto

Estos objetos pueden insertarse en las páginas de los libros en modo de proyecto mediante el IDE de HAEduc. Mediante el script estos objetos visuales pueden ser instanciados mediante el empleo de clases internas de Sora Script . A cada objeto le corresponde una clase cuyo nombre está formado por el prefijo SS seguido del nombre del objeto:

1. SSImagen
2. SSBotón
3. SSEtiqueta

4. SSCaja de texto
5. SShtml
6. SSVideo
7. SSPolígono
8. SSLista
9. SSReloj
10. SSTabla
11. SSTexto

De esta forma por ejemplo para crear una lista mediante script podría hacerse de esta forma:

```
Dim milista = SSLista("lista1",sspunto(100,100),ssDim(80,200))
```

El nombre de la lista es empleado posteriormente para la captura de los eventos de este objeto.

Al mismo tiempo estos objetos pertenecen a una clase superior de la cual heredan muchas de sus propiedades y métodos, esta clase es SSOBJETO. Es por ello que los métodos comunes de todos los objetos (como el de establecer la posición o la visibilidad) existen en cada objeto por haberlos heredados de SSOBJETO.

Clase SSOBJETO

ssobjeto::ssobjeto

ssobjeto() as ssobjeto

Instancia de un objeto genérico.

ssobjeto::estposicion

estposicion(pos as **ssPunto**)

Establece la posición del objeto.

ssobjeto::obtposicion

obtposicion() as **ssPunto**

Devuelve la posición del objeto

ssobjeto::obtposicionx

obtposicionx() as **integer**

Devuelve la coordenada x de la posición del objeto.

ssobjeto::obtposiciony

obtposiciony() as **integer**

Devuelve la coordenada y de la posición del objeto.

ssobjeto::rotar

rotar(angulo as **integer**)

Rota el objeto (si el tipo de objeto lo permite) hasta el ángulo especificado.

ssobjeto::obtangulo

obtangulo() as **integer**

Devuelve el ángulo de rotación del objeto.

ssobjeto::mostrar

mostrar()

Hace visible el objeto.

ssobjeto::ocultar

oocultar ()
Ocultar el objeto.

ssobjeto::estvisible

estvisible([visible=1 as **integer**])
Establece la visibilidad del objeto.

ssobjeto::obtvisible

obtvisible() as **integer**
Devuelve 1 si el objeto está visible, de lo contrario devuelve 0.

ssobjeto::estarrastrado

estarrastrado(arrastrado as **integer**)
Permite el arrastrado *del objeto*.

ssobjeto::estdimensiones

estdimensiones(dimension as **ssDim**)
Establece las dimensiones del objeto.

ssobjeto::estpuntero

estpuntero(puntero as **integer**)
Establece el *puntero* del mouse para *este objeto*. Los punteros disponibles son:

SS_PUNTERO_MANO	Representa una mano
SS_PUNTERO_EDICION	Puntero de edición de texto
SS_PUNTERO_PROHIBIDO	Símbolo de prohibición
SS_PUNTERO_INTERROGACION	Símbolo de interrogación
SS_PUNTERO_INTERROGACION	Flecha con dirección norte-sur y este-oeste
SS_PUNTERO_NS	Flecha con dirección norte-sur
SS_PUNTERO_NOSE	Flecha con dirección noroeste-sureste
SS_PUNTERO_EO	Flecha con dirección este-oeste
SS_PUNTERO_CRUZ	Símbolo en forma de cruz
SS_PUNTERO_ESPERA	Símbolo de espera
SS_PUNTERO_FLECHAESPERA	Símbolo de espera con flecha
SS_PUNTERO_FLECHA	Símbolo flecha (puntero normal)
SS_PUNTERO_IZQUIERDO	Símbolo flecha invertida
SS_PUNTERO_DERECHO	Símbolo flecha horizontal hacia la derecha

ssobjeto::obtpuntero

obtpuntero() as **integer**
Devuelve el puntero del mouse del objeto.

ssobjeto::estimgenpuntero

estimgenpuntero(imagen as **string**)
Establece la imagen que servirá de puntero del mouse *para este objeto*.

ssobjeto::destruir

destruir()
Destruye el objeto liberando la memoria ocupada por este.

ssobjeto::estnombre

estnombre(nuevonombre as string)
Cambia el nombre del objeto por *nuevonombre*.

ssobjeto::obtdimensiones

obtdimensiones() as **ssDim**
Devuelve las dimensiones del objeto.

ssobjeto::obtancho

obtancho() as **integer**
Devuelve el ancho de un objeto.

ssobjeto::obtaltura

obtaltura() as **integer**
Devuelve la altura de un objeto.

ssobjeto::estopacidad

estopacidad(opc as **integer**)
Establece la opacidad al valor *opc*.
(Solo para Windows)

Clase SSTabla

Hereda de SSOBJETO.

ssttabla::ssttabla

ssttabla(nombre as **string**, columnas as **integer**, filas as **integer**, pos as **ssPunto**, dimensiones as **ssDim**) as **ssttabla**

Crear una tabla en el proyector, donde:

<i>Nombre</i>	Nombre del objeto en el proyector
<i>columnas</i>	Número de columnas
<i>filas</i>	Número de filas
<i>pos</i>	posición de la tabla
<i>dimensiones</i>	dimensiones de la tabla

Nota: El número máximo de filas y columnas en 50.

ssttabla::adicionarcolumnas

adicionarcolumnas(ncol as **integer**)
Adiciona *ncol* columnas a una tabla.

ssttabla::adicionarfilas

adicionarfila(nfil as **integer**)
Adiciona *nfil* filas a una tabla.

ssttabla::eliminarcolumna

eliminarcolumna(scolumna as **integer**)
Elimina la columna *scolumna* una tabla.

::eliminarfila

eliminarfila(sfila as **integer**)
Elimina la fila *sfila* de una tabla

ssttabla::estedicion

estedicion(edit as **integer**)
Hará editable o no en tiempo de ejecución (según el valor 0 o 1 del parámetro *edit*) una tabla.

ssttabla::vaciar

vaciar(nombretabla as **string**)
Elimina el contenido de una tabla.

ssttabla::estvalorcelda

estvalorcelda(scolumna as **integer**, sfila as **integer**, valor as **string**)
Establece el *valor* de una celda localizada en la columna *scolumna* y la fila *sfila* de una tabla.

ssttabla::estcolorlinea

estcolorlinea(color as **ssColor**)
Establece el color de la línea de una tabla.

ssttabla::estfuentecelda

estfuentecelda(scolumna as **integer**, sfila as **integer**, fuente as **ssFuente**)
Establece la fuente de una celda localizada en la columna *scolumna* y la fila *sfila* de una tabla.

sstable::estfuente

estfuente(fuente as **ssFuente**)
Establece la fuente de todas las celdas de una tabla.

sstable::estcolorfuente

estcolorfuente(color as **ssColor**)
Establece el color de la fuente de una tabla.

sstable::estcolorfuentecelda

estcolorfuentecelda(sfila as integer, scolumna as integer, color as **ssColor**)
Establece el color de la celda localizada en la columna *scolumna* y la fila *sfila* de una tabla

sstable::estcolorfondocelda

estcolorfondocelda(scolumna as integer, sfila as integer, color as **ssColor**)
Establece el color de fondo de la celda localizada en la columna *scolumna* y la fila *sfila* de una tabla.

:: sstable::estcolorfondo

estcolorfondo(rcolor as **ssColor**)
Establece el color de fondo de todas las celdas de una tabla.

sstable::estdimensioncolumna

estdimensioncolumna(scolumna as **integer**, tam as **integer**)
Establece el tamaño *tam* de la columna *scolumna* de una tabla.

:: sstable::estdimensionfila

estdimensionfila (sfila as **integer**, tam as **integer**)
Establece el tamaño *tam* de la fila *sfila* de una tabla.

sstable::obtfuentecelda

obtfuentecelda(scolumna as **integer**, sfila as **integer**) as **ssFuente**
Devuelve la fuente de la celda localizada en la columna *scolumna* y la fila *sfila* de la tabla.

sstable::obtcolorfuentecelda

obtcolorfuentecelda(scolumna as **integer**, sfila as **integer**) as **ssColor**
Devuelve el color de fuente de una celda localizada en la columna *scolumna* y la fila *sfila* de la tabla.

sstable::obtnumerocolumnas

obtnumerocolumnas() as **integer**
Devuelve el número de columnas de una tabla.

sstable::obtnumerofilas

obtnumerofilas () as **integer**
Devuelve el número de filas de una tabla.

sstable::obtdimensioncolumna

obtdimensioncolumna(scolumna as **integer**) as **integer**
Devuelve el tamaño de la columna *scolumna* de una tabla.

sstable::obtdimensionfila

obtdimensionfila (sfila as **integer**) as **integer**
Devuelve el tamaño de la fila *sfila* de una tabla.

sstable::obtcolorlinea

obtcolorlinea() as **ssColor**
Devuelve el color línea de una tabla cuyo nombre es *nombretabla*. Si existe más de una tabla con dicho nombre, se tomará en cuenta sólo la primera según el orden de creación.

sstable::obtvalorcelda

obtvalorcelda(sfila as **integer**, scolumna as **integer**) as **string**
Devuelve el valor de una celda localizada en la columna *scolumna* y la fila *sfila* de una tabla.

Clase SSBoton

Hereda de SSOBJeto.

ssboton::ssboton

ssboton(nombre as **string**, fimagen as **string**, pos as **ssPunto**, dimensiones as **ssDim**) as **ssboton**

Crea un objeto botón en el proyector a partir de un fichero de imagen *fimagen*.

ssboton::estimagen1

estimagen1(fimagen as **string**)

Establece la primera imagen del botón a partir de un fichero de imagen.

ssboton::estimagen1IB

estimagen1IB(data as **ssImagenBase**)

Establece la primera imagen del botón a partir de un objeto ssImagenBase.

ssboton::estimagen2

estimagen2(fimagen as **string**)

Establece la segunda imagen del botón a partir de un fichero de imagen.

ssboton::estimagen2IB

estimagen2IB(data as **ssImagenBase**)

Establece la segunda imagen del botón a partir de un objeto ssImagenBase.

ssboton::estimagen3

estimagen3(fimagen as **string**)

Establece la tercera imagen del botón a partir de un fichero de imagen.

ssboton::estimagen3IB

estimagen3IB(data as **ssImagenBase**)

Establece la tercera imagen del botón a partir de un objeto ssImagenBase.

ssboton::esttexto

esttexto(texto as **string**)

Establece el *texto* de la etiqueta del botón.

ssboton::estfuente

estfuente(fuente as **ssFuente**)

Establece la fuente de la etiqueta del botón.

ssboton::obttexto

obttexto() as **string**

Devuelve el texto de la etiqueta del botón.

ssboton::obtfuente

obtfuente() as **ssFuente**

Devuelve la fuente del botón.

ssboton::obtcolor1

obtcolor1() as **ssColor**

Devuelve el primer color de la etiqueta del botón.

ssboton::obtcolor2

obtcolor2() as **ssColor**

Devuelve el segundo color de la etiqueta del botón.

ssboton::obtcolor3

obtcolor3() as **ssColor**

Devuelve el tercer color de la etiqueta del botón.

ssboton::estcolor1

`estcolor1(color as ssColor)`
Establece el primer color de la etiqueta del botón.

ssboton::estcolor2
`estcolor2(color as ssColor)`
Establece el segundo color de la etiqueta del botón.

ssboton::estcolor3
`estcolor3(color as ssColor)`
Establece el tercer color de la etiqueta del botón.

ssboton::estangulocolores
`estangulocolores(angulo as integer)`
Establece el ángulo de rotación de los colores originales de la imagen del objeto permitiéndole mostrar diversas apariencias. Los valores 0, 50 y 100 mantienen la imagen del objeto sin cambios siendo 0 y 100 los límites superior e inferior.

Clase SSEtiqueta

Hereda de SSOBJETO.

ssetiqueta::ssetiqueta
`ssetiqueta(nombre as string, texto as string, pos as ssPunto) as ssEtiqueta`
Crea un objeto etiqueta

ssetiqueta::estcolor1
`estcolor1(color as ssColor)`
Establece el primer color de la etiqueta.

ssetiqueta::estcolor2
`estcolor2(color as ssColor)`
Establece el segundo color de la etiqueta.

ssetiqueta::estcolor3
`estcolor3(color as ssColor)`
Establece el tercer color de la etiqueta.

ssetiqueta::esttexto
`esttexto(texto as string)`
Establece el texto de la etiqueta.

ssetiqueta::obtttexto
`obtttexto() as string`
Devuelve el texto de la etiqueta.

ssetiqueta::estfuente
`estfuente(fuente as ssFuente)`
Establece la fuente de la etiqueta.

ssetiqueta::obtfuente
`obtfuente() as ssFuente`
Devuelve la fuente de la etiqueta.

ssetiqueta::obtcolor1
`obtcolor1() as ssColor`
Devuelve el primer color de la etiqueta.

ssetiqueta::obtcolor2
`obtcolor2() as ssColor`
Devuelve el segundo color de la etiqueta.

ssetiqueta::obtcolor3

obtcolor3() as **ssColor**
Devuelve el tercer color de la etiqueta.

Clase SSPoligono

Hereda de **SSObjeto**.

sspoligono::sspoligono

sspoligono(nombre as **string**) as **sspoligono**
Crea un polígono en el proyector.

sspoligono::adicionarpunto

adicionarpunto(punto as **ssPunto**)
Adiciona un punto al polígono.

sspoligono::insertarpunto

insertarpunto(pos as **integer**, punto as **ssPunto**)
Inserta un punto en la posición *pos* de lista de puntos del polígono.

sspoligono::eliminarpunto

eliminarpunto(pos as **integer**)
Elimina el punto perteneciente a la posición *pos* de lista de puntos del polígono.

sspoligono::estestilo

estestilo(estilo as **integer**)
Establece el estilo de relleno del polígono. El parámetro *estilo* puede tomar los siguientes valores:

SS_POLIGONO_RELLENO_SUPERPUESTO
SS_POLIGONO_RELLENO_SOLAPADO

sspoligono::estestilolinea

estestilolinea (estilo as **integer**)
Establece el estilo de línea del polígono. El parámetro *estilo* puede tomar los siguientes valores:

SS_EL_SOLIDO
SS_EL_PUNTEADO
SS_EL_LLINEAS
SS_EL_CLINEAS
SS_EL_LINEASPUNTOS
SS_EL_USUARIO
SS_EL_TRANSPARENTE

sspoligono::estentramado

estentramado (entramado as **integer**)
Establece el entramado de relleno del polígono. El parámetro *entramado* puede tomar los siguientes valores:

SS_PE_DIAGONAL
SS_PE_CRDIAGONAL
SS_PE_FDIAGONAL
SS_PE_CR
SS_PE_HORIZONTAL
SS_PE_VERTICAL
SS_PE_SOLIDO
SS_PE_TRANSPARENTE

sspoligono::estcolorfondo

`estcolorfondo(color as ssColor)`
Establece el color de fondo del polígono.

sspoligono::estcolorlinea
`estcolorlinea (color as ssColor)`
Establece el color de línea del polígono.

sspoligono::estgrosorlinea
`estgrosorlinea(grosor as integer)`
Establece el grosor de línea del polígono.

sspoligono::estpunto
`estpunto(punto as integer, punto as ssPunto)`
Establece el valor de las coordenadas de un *punto* de la lista de puntos del polígono.

sspoligono::obtpunto
`obtpunto(punto as integer) as ssPunto`
Devuelve las coordenadas de un *punto* de la lista de puntos del polígono.

Clase SSCtexto

Caja de texto.

Hereda de SObjeto.

ssctexto::ssctexto
`ssctexto(nombre as string, pos as ssPunto, largo as integer) as ssctexto`
Crea una caja de texto en el proyector.

ssctexto::esttexto
`esttexto(texto as string)`
Establece el texto de la caja de texto.

ssctexto::estfuente
`estfuente(fuente as ssFuente)`
Establece la fuente de la caja de texto.

ssctexto::vacía
`vacía() as integer`
Devuelve 1 si el objeto caja de texto se encuentra vacío. De lo contrario devuelve 0.

ssctexto::vaciar
`vaciar()`
Elimina el texto de la caja de texto.

ssctexto::estcolormarco
`estcolormarco (color as ssColor)`
Establece el color del marco de la caja de texto.

ssctexto::estmarcovisible
`estmarcovisible (visible as integer)`
Establece la visibilidad del marco de la caja de texto.

ssctexto::obttexto
`obttexto() as string`
Devuelve el texto de la caja de texto.

ssctexto::obtfuente
`obtfuente() as ssFuente`
Devuelve la fuente de la caja de texto.

ssctexto::obtcolormarco

obtcolormarco () as **ssColor**
Devuelve el color del marco de la caja de texto.

Clase **SSImagen**

Objeto imagen.

Hereda de SSObjeto.

ssimagen::ssimagen

ssimagen(nombre as **string**, fichero as **string**, pos as **ssPunto**, dimension as **ssDim**) as **ssimagen**

Crea una imagen en el proyector.

ssimagen::estimagen

estimagen(fichero as **string**)

Establece la imagen del objeto a partir de un fichero de imagen.

ssimagen::estimagenIB

estimagenIB(data as **ssImagenBase**)

Establece la imagen del objeto a partir de un objeto **ssImagenBase**.

ssimagen::estimagen2

estimagen2(fichero as **string**)

Establece la imagen de intercambio del objeto a partir de un fichero de imagen.

ssimagen::estimagen2IB

estimagen2IB(data as **ssImagenBase**)

Establece la imagen de intercambio del objeto a partir de un objeto **ssImagenBase**.

ssimagen::esttipoimagen

esttipoimagen(tipo as **integer**)

Establece el tipo del objeto **ssimagen**, donde *tipo* puede tomar los valores:

SS_IMAGEN_NORMAL	imagen normal
SS_IMAGEN_INTERCAMBIO_OVER	imagen de intercambio generado por el evento MouseOver
SS_IMAGEN_INTERCAMBIO_DOWN	imagen de intercambio generado por el evento MouseDown

ssimagen::obttipoimagen

obttipoimagen() as **integer**

Devuelve el tipo de un objeto **Imagen**. El valor devuelto puede ser:

SS_IMAGEN_NORMAL	imagen normal
SS_IMAGEN_INTERCAMBIO_OVER	imagen de intercambio generado por el evento MouseOver
SS_IMAGEN_INTERCAMBIO_DOWN	imagen de intercambio generado por el evento MouseDown

ssimagen::obtestado

obtestado() as **integer**

Devuelve el estado actual de la imagen si el objeto es del tipo de imagen **SS_IMAGEN_INTERCAMBIO_DOWN** (ver **esttipoimagen**).

ssimagen::estestado

estestado(estado as **integer**)

Establece el estado del objeto si este es del tipo de imagen **SS_IMAGEN_INTERCAMBIO_DOWN** (ver **esttipoimagen**). El parámetro estado puede tomar los valores 0 y 1.

ssimagen::estangulocolores

estangulocolores(angulo as **integer**)

Establece el ángulo de rotación de los colores originales de la imagen del objeto permitiéndole mostrar diversas apariencias. Los valores 0, 50 y 100 mantienen la imagen del objeto sin cambios siendo 0 y 100 los límites superior e inferior.

Clase **SShtml**

Objeto html.

Hereda de **SSObjeto**.

sshtml::sshtml

sshtml(nombre as **string**, fichero as **string**, pos as **ssPunto**, dimension as **ssDim**) as **sshtml**

Crea un objeto html en el proyector y carga un *fichero* existente. El parámetro fichero puede establecerse inicialmente vacío (""), si no se desea cargar fichero alguno en la construcción del objeto.

sshtml::estfichero

estfichero(fichero as **string**)

Establece el fichero e intenta cargarlo.

sshtml::estpagina

estpagina(fichero as **string**)

Establece el fichero e intenta cargarlo

sshtml::adicionarcodigo

adicionarcodigo(codigohtml as **string**)

Agrega código html para ser interpretado por el objeto.

Ejemplo:

```
dim objeto = new sshtml("mihtml", "", sspunto(10,10), ssdim(300,200))
```

```
objeto.adicionarcodigo("<html><body>Hola, mundo</body></html>")
```

sshtml::estcodigo

estcodigo(codigohtml as **string**)

Establece el código html para ser interpretado por el objeto.

sshtml::obtpagina

obtpagina() as **string**

Devuelve el nombre del fichero cargado actualmente del objeto.

sshtml::obttitulopagina

obttitulopagina() as **string**

Devuelve el nombre interno de la página cargada actualmente.

sshtml::atras

atras()

Provoca una navegación hacia atrás (si es posible).

sshtml::adelante

adelante()

Provoca una navegación hacia adelante (si es posible).

sshtml::puedeadelante

puedeadelante() as **integer**

Devuelve 1 si el objeto puede navegar hacia delante de lo contrario devuelve 0.

sshtml::puedeatras

puedeatras() as **integer**

Devuelve 1 si el objeto puede navegar hacia atrás de lo contrario devuelve 0.

sshtml::borrarhistorial

`borrarhistorial()`
Borra el historial de navegación del objeto.

sshtml::seleccionartodo

`seleccionartodo()`
Selecciona todo el contenido.

sshtml::obttextoseleccionado

`obttextoseleccionado()` as **string**
Devuelve el texto seleccionado en el objeto.

sshtml::desconectarvinculos

`desconectarvinculos(des as integer)`
El objeto `Html` permite cargar páginas que contengan vínculos a otras páginas. Al hacer clic sobre uno de estos vínculos el objeto realizará dos acciones, la primera será generar el evento `Linkclick` y la segunda, tratar de navegar a la dirección de este vínculo. Si se necesita que el objeto sólo responda al evento sin intentar navegar, entonces este método permite desconectar o reconectar dicha navegación.

Clase SSLista

Objeto Lista.

Hereda de SObjeto.

sslista::sslista

`sslista(nombre as string, pos as ssPunto, dimension as ssDim [, estilo as integer = SS_LISTA_NORMAL])` as **sslista**

Crea un objeto lista en el proyector.

El parámetro `estilo` puede tener los valores:

<code>SS_LISTA_NORMAL</code>	Lista normal
<code>SS_LISTA_ORDENADA</code>	Elementos ordenados
<code>SS_LISTA_EXTENDIDA</code>	Múltiple selección
<code>SS_LISTA_EXTENDIDA_ORDENADA</code>	Múltiple selección y ordenada

sslista::adicionarelemento

`adicionarelemento(elemento as string)`
Añade un nuevo elemento a la lista.

sslista::insertarelemento

`sslistainsertarelemento(elemento as string, pos as integer)`
Inserta un nuevo elemento en la posición `pos` de la lista.

sslista::eliminar elemento

`eliminar elemento(pos as integer)`
Elimina el elemento situado en la posición `pos` de la lista.

sslista::esttexto

`estelemento(texto as string, pos as integer)`
Establece el texto del elemento situado en la posición `pos` de la lista.

sslista::vaciar

`vaciar()`
Elimina todos los elementos de la lista.

sslista::vacía

`vacía()` as **integer**
Devuelve si está vacía la lista.

sslista::estcolorfuente

`estcolorfuente(color as ssColor)`
Establece el color de fuente de la lista.

sslista::estfuente

estfuente(fuente as **ssFuente**)
Establece la fuente de la lista.

sslista::estcolorfondo

estcolorfondo(color as **ssColor**)
Establece el color de fondo de la lista.

sslista::esttooltip

esttooltip(texto as **string**)
Establece el texto de información (tooltip) de la lista.

sslista::obtseleccion

obtseleccion() as **string**
Devuelve una cadena con los índices de los elementos seleccionados en la lista.

sslista::obtextoseleccionado

obtextoseleccionado() as **string**
Devuelve una cadena con los valores de los elementos seleccionados en una lista.

sslista::obttooltip

obttooltip() as **string**
Devuelve el texto de información (tooltip) de la lista.

sslista::llenar

llenar(sep as **string**, elementos as **string**)
Permite establecer una lista de elementos que se encuentren separados por un caracter de separación.
Ejemplo:

```
dim lista = new sslista("lista",sspunto(10,10),ssdim(100,100))
elementos="uno;dos;tres"
lista.llenar(";", elementos)
```

sslista::obtcolorfondo

obtcolorfondo() as **ssColor**
Devuelve el color de fondo de la lista.

sslista::obtfuente

obtfuente() as **ssFuente**
Devuelve la fuente la lista.

sslista::obtcolorfuente

obtcolorfuente() as **ssColor**
Devuelve el color de la fuente.

sslista::obtelemento

obtelemento(pos as **integer**) as **string**
Devuelve el texto del elemento situado en la posición *pos* de la lista.

sslista::obtseleccion

obtseleccion() as **string**
Devuelve una cadena con los índices de los elementos seleccionados (separados por un \n) en la lista.

sslista::contar

contar() as **integer**
Devuelve el número de elementos que contiene la lista.

sslista::vacía

vacía() as **integer**
Devuelve 1 si una lista está vacía de lo contrario devuelve 0.

sslista::estseleccion

estseleccion(elemento as **integer**)
Establece el *elemento* seleccionado.

sslista::buscar

buscar(texto as **string**) as **integer**
Busca un texto entre los elementos de una lista y devuelve su posición si lo encuentra, de no ser así devuelve -1.

Clase SSReloj

Objeto Reloj.

Hereda de SSOBJETO.**ssreloj::ssreloj**

ssreloj(nombre as **string**) as **ssreloj**
Crea un objeto Reloj en el proyector. Este objeto no es visible en modo de ejecución.

ssreloj::iniciar

iniciar()
Inicia el funcionamiento del reloj.

ssreloj::detener

detener()
Detiene el funcionamiento del reloj.

ssreloj::estintervalo

estintervalo(intervalo as **integer**)
Establece el intervalo del reloj. El parámetro intervalo está dado en milisegundos (1000 significa 1 segundo).

ssreloj::obtintervalo

obtintervalo() as **integer**
Devuelve el intervalo de un reloj.

ssreloj::iniciado

iniciado() as **integer**
Devuelve 1 si un reloj está funcionando, de no ser así devuelve 0.

Clase SSVideo

Objeto Video.

Hereda de SSOBJETO.**ssvideo::ssvideo**

ssvideo(nombre as **string**, fichero as **string**, pos as **ssPunto**, dimension as **ssDim**) as **ssvideo**
Crea un objeto Video en el proyector e intenta cargar un *fichero*. Si no se desea que se cargue un fichero en la creación entonces este parámetro se puede pasar con valor "".

ssvideo::reproducir

reproducir()
Inicia la reproducción del video.

ssvideo::pausar

pausar()
Pone en pausa la reproducción del video.

ssvideo::cargar

cargar(fichero as **string**) as **integer**

Carga un *fichero* de video y devuelve 1 si realiza esta operación con éxito.

ssvideo::posicionar

posicionar(pos as **integer**)

Posiciona la reproducción en *pos* del video.

ssvideo::esttasareproduccion

esttasareproduccion(tasa as **integer**)

Establece la tasa (velocidad) de reproducción del video.

ssvideo::estvolumen

estvolumen(vol as **number**)

Establece el volumen del video. El rango válido es de 0 a 1.

ssvideo::mostrarcontroles

mostrarcontroles(estilo as **integer**)

Visualiza los controles en diferentes estilos del objeto video. El parámetro estilo puede tomar los siguientes valores:

SS_CONTROLES_NO	Sin controles
SS_CONTROLES_AVANCE	Muestra la barra de posición
SS_CONTROLES_VOLUMEN	Muestra los controles de volumen
SS_CONTROLES_PORDEFECTO	Estado por defecto del sistema

Nota: Este método sólo funciona bajo plataforma Windows.

ssvideo::detener

detener()

Detiene la reproducción del video.

ssvideo::obtposicion

obtposicion() as **integer**

Devuelve la posición de reproducción de un video.

ssvideo::obtdimensionesoptimas

obtdimensionesoptimas() as **ssDim**

Devuelve las dimensiones óptimas para la reproducción de un fichero de video previamente cargado de un video.

ssvideo::obttasareproduccion

obttasareproduccion() as **number**

Devuelve la tasa (velocidad) de reproducción del video.

ssvideo::obtvolumen

obtvolumen() as **number**

Devuelve el volumen de un video.

ssvideo::obtestado

ssvideoobtestado() as **integer**

Devuelve el estado actual de un video. Este método puede devolver los siguientes valores:

SS_DETENIDO	El video está detenido
SS_PAUSA	El video está en pausa
SS_REPRODUCIENDO	El video se está reproduciendo

ssvideo::obtduracion

obtduracion() as **integer**

Devuelve la duración del video cargado en milisegundos.

Clase SSTexto

Objeto Texto.

Hereda de SObjeto.**sstexto::sstexto**

sstexto(nombre as **string**, pos as **ssPunto**, dimension as **ssDim**) as **sstexto**
Crea un objeto texto en el proyector.

sstexto::adicionartexto

adicionartexto(texto as **string**)
Añade un texto al contenido del objeto.

sstexto::vaciar

vaciar()
Elimina el contenido del objeto.

sstexto::descartarediccion

descartarediccion()
Descarta la edición y establece el estado del objeto como si no se hubiera editado.

sstexto::editable

editable(editable as **integer**)
Establece la posibilidad de edición del objeto en tiempo de ejecución.
el parámetro *editable* puede tomar los valores 0 o 1.

sstexto::estpuntoinsercion

estpuntoinsercion(pos as **integer**)
Establece el punto de inserción de texto a la posición *pos*.

sstexto::estpuntoinsercionfinal

estpuntoinsercionfinal()
Establece el punto de inserción de texto al final del contenido.

sstexto::estmaximalongitudtexto

estmaximalongitudtexto(max as **integer**)
Establece la máxima cantidad de caracteres que admitirá el objeto.

sstexto::estseleccion

estseleccion(posinicial as **integer**, posfinal as **integer**)
Selecciona un fragmento de texto desde una posición inicial (posinicial) a una final (posfinal).

sstexto::esttexto

esttexto(texto as **string**)
Establece el texto sin marcar al objeto como modificado.

sstexto::cambiar texto

cambiar texto(texto as **string**)
Cambia el texto marcando al objeto como modificado.

sstexto::mostrarposicion

mostrarposicion(pos as **integer**)
Procura visualizar el carácter situado en la posición *pos* en la ventana del objeto.

sstexto::escribir

escribir(texto as **string**)
Inserta un *texto* en el punto de inserción actual.

sstexto::deshacer

deshacer()
Ejecuta el comando deshacer.

sstexto::rehacer

rehacer()
Ejecuta el comando rehacer.

sstexto::guardarguardar(fichero as **string**)Guarda el contenido de un objeto texto en un *fichero*.**sstexto::reemplazar**reemplazar(posini as **integer**, posfin as **integer**, valor as **string**)Reemplaza el texto situado desde la posición posini hasta posfin por un *valor*.**sstexto::pegar**

pegar()

Ejecuta el comando pegar.

sstexto::copiar

copiar()

Ejecuta el comando copiar.

sstexto::cortar

cortar()

Ejecuta el comando cortar.

sstexto::cargarcargar(fichero as **string**)Carga el contenido de un *fichero* en el objeto.**sstexto::estcolortextoseleccion**estcolortextoseleccion(posini as **integer**, posfin as **integer**, color as **ssColor**)

Establece el color del texto desde la posición posini hasta posfin.

sstexto::estcolorfondoseleccionestcolorfondoseleccion(posini as **integer**, posfin as **integer**, color as **ssColor**)

Establece el color de fondo del texto desde la posición posini hasta posfin.

sstexto::estfuenteseleccionestfuenteseleccion(posini as **integer**, posfin as **integer**, fuente as **ssFuente**)

Establece la fuente del texto desde la posición posini hasta posfin.

sstexto::estalineacionseleccionestalineacionseleccion(posini as **integer**, posfin as **integer**, alineacion as **integer**)Establece la alineacion del texto desde la posición posini hasta posfin. El parámetro de *alineación* puede tomar los siguientes valores:

SSTEXTO_ALINEACION_PORDEFECTO	Por defecto
SSTEXTO_ALINEACION_IZQUIERDA	Izquierda
SSTEXTO_ALINEACION_CENTRADA	Centrada
SSTEXTO_ALINEACION_DERECHA	Derecha
SSTEXTO_ALINEACION_JUSTIFICADA	Justificada

sstexto::puedecopiarpuedecopiar() as **integer**

Devuelve si un objeto texto puede ejecutar el comando copiar.

sstexto::puedecortarpuedecortar() as **integer**

Devuelve si un objeto texto puede ejecutar el comando cortar.

sstexto::puedepegarpuedepegar() as **integer**

Devuelve si un objeto texto puede ejecutar el comando pegar.

sstexto::puederehacerpuederehacer() as **integer**

Devuelve si un objeto texto puede ejecutar el comando rehacer.

sstexto::puededeshacer

puededeshacer() as **integer**

Devuelve si un objeto texto puede ejecutar el comando deshacer.

sstexto::obtpuntoinsercion

obtpuntoinsercion() as **integer**

Devuelve el punto de inserción en el contenido de un objeto texto.

sstexto::obtlongitudlinea

obtlongitudlinea(línea as **integer**) as **integer**

Devuelve el número de caracteres en una *línea* de un objeto texto.

sstexto::obttextolinea

obttextolinea(línea as **integer**) as **string**

Devuelve el contenido de una *línea* de un objeto texto.

sstexto::contarlineas

sstextocontarlineas() as **integer**

Devuelve el número de líneas de un objeto texto.

sstexto::obtseleccion

obtseleccion() as **string**

Devuelve el texto seleccionado de un objeto texto.

sstexto::obtvalor

obtvalor() as **string**

Devuelve el contenido de un objeto texto.

sstexto::obtfuente

obtfuente() as **ssFuente**

Devuelve la fuente del objeto.

Bases de datos

En la actual versión de HAEDUC sólo es posible el trabajo con bases de datos de SQLite. Para el manejo de estas bases de datos se emplea la clase `ssBaseDatos` y para la gestión de los resultados de consultas la clase `ssResultado`.

Clase ssBaseDatos

ssBaseDatos::ssBaseDatos

ssBaseDatos(fichero as **string**) as **ssBaseDatos**

Abre una base de datos existente en un fichero. El parámetro `fichero` puede enviarse vacío si se desea construir un objeto `SSBaseDatos` sin abrir previamente una base de datos.

ssBaseDatos::consultaconresultado

consultaconresultado(consulta as **string**) as **ssResultado**

Ejecuta una consulta `sqlite` y devuelve el resultado en un objeto de tipo `ssResultado`.

ssBaseDatos::abrir

abrir(fichero as **string**) as **integer**

Intenta abrir una base de datos existente en un fichero.

ssBaseDatos::cerrar

cerrar()

Cierra una base de datos.

ssBaseDatos::abierta

abierta() as **integer**

Devuelve 1 si una base de datos está abierta, sino devuelve 0.

ssBaseDatos::atras

atras()
 Deshace los cambios en una base de datos.

ssBaseDatos::confirmar

confirmar()
 Envía un comando para que se establezcan los cambios realizados en una base de datos.

ssBaseDatos::consulta

consulta(consulta as **string**)
 Ejecuta una consulta sqlite en una base de datos.

ssBaseDatos::existetabla

existetabla(tabla as **string**) as **integer**
 Devuelve 1 si existe una tabla en la base de datos. De lo contrario devuelve 0.

Clase ssResultado

Clase para la gestión de los resultados de consultas mediante el método ConsultaConResultado de la clase ssBaseDatos.

ssResultado::ssResultado

ssResultado() as **ssResultado**
 Crea un objeto ssResultado.

ssResultado::proximo

proximo() as **integer**
 Si existe un próximo elemento en la lista de resultados salta a este y devuelve 1, de no ser así devuelve 0.

ssResultado::resultadoscc

resultadoscc(campo as **string**) as **ssCC**
 Devuelve el contenido del elemento actual para un campo en la lista de resultados, como una cadena de caracteres de tipo ssCC.

ssResultado::resultadoentero

resultadoentero(campo as **string**) as **integer**
 Devuelve el contenido del elemento actual para un campo en la lista de resultados, como número entero.

ssResultado::resultadonumber

resultadonumber(campo as **string**) as **number**
 Devuelve el contenido del elemento actual para un campo en la lista de resultados, como un number.

ssResultado::camponulo

camponulo(campo as **string**) as **integer**
 Devuelve 1 si un campo es nulo en el elemento actual.

Ejemplo del empleo de las clases para gestión de bases de datos:

```
//Se supone la existencia de una base de datos en /home/prueba.db
// que previamente tenga una tabla nombrada datos con
// el campo nombre y el campo apellido.
```

```
dim base =new ssbasedatos("")
base.abrir("/home/prueba.db")
```

```
//inserta valores mediante una consulta
base.consulta("INSERT INTO datos (nombre, apellidos) VALUES ('juan', 'fernandez');")
```

```
//objeto para almacenar los resultados de consultas
dim result= new ssresultado()

//almacenamos en el resultado
result=base.consultaconresultado("SELECT * FROM datos")

dim str as string
dim ss as ssc

while result.proximo()
    //obtiene el valor actual para el campo nombre
    ss=result.resultadoscc("nombre")
    str = str & "\n" & ss.string()
end while

ssMensaje(str)

base.cerrar()
```

Captura de eventos

Las aplicaciones desarrolladas con HAEduc son guiadas por eventos. Por tanto todas las acciones que se ejecuten serán acciones derivadas de la ocurrencia de un evento. Es por ello que es necesario conocer la sintaxis de los eventos y los parámetros que manejan cada uno de ellos.

En general un evento se puede capturar mediante una subrutina cuyo nombre esté compuesto por el nombre del evento y el del objeto sobre el que ocurre dicho evento (o la aplicación).

Sintaxis:

```
SUB nombreevento_nombreobjeto (listaparametros)
```

```
END SUB
```

```
//ejemplo del evento mouseclick sobre el objeto figura
Sub mouseclick_figura()
ssMensaje("Ocurrió el evento clic")
end sub
```

A continuación se muestran los eventos que pueden ser capturados en las aplicaciones de HAEduc.

Eventos del ratón sobre los objetos

Nombre: Mousedown

Descripción: Ocurre al presionar el botón del mouse sobre un objeto

Sintaxis: Sub Mousedown_objeto(p1)

Parámetros:

p1: contiene el botón del mouse que generó el evento

Nombre: Mouseup

Descripción: Ocurre al liberar el botón del mouse sobre un objeto

Sintaxis: Sub Mouseup_objeto(p1)

Parámetros:

p1: contiene el botón del mouse que generó el evento

Nombre: Mousedc

Descripción: Ocurre al hacer doble clic sobre un objeto

Sintaxis: Sub Mousedc_objeto(p1)

Parámetros:

p1: contiene el botón del mouse que generó el evento

Nombre: Mouseclick

Descripción: Ocurre al hacer clic sobre un objeto

Sintaxis: Sub Mouseclick_objeto(p1)

Parámetros:

p1: contiene el botón del mouse que generó el evento

Nombre: Mousedragging

Descripción: Ocurre al arrastrar sobre un objeto

Sintaxis: Sub Mousedragging_objeto(p1,p2)

Parámetros:

p1: contiene la coordenada x del mouse

p2: contiene la coordenada y del mouse

Nombre: Mouseover

Descripción: Ocurre al mover el puntero sobre un objeto

Sintaxis: Sub Mouseover_objeto(p1,p2)

Parámetros:

p1: contiene la coordenada x del mouse

p2: contiene la coordenada y del mouse

Nombre: Mousein

Descripción: Ocurre al entrar el puntero del mouse en un objeto

Sintaxis: Sub Mousein_objeto()

Parámetros:

Sin parámetros

Nombre: Mouseout

Descripción: Ocurre al salir el puntero del mouse de un objeto

Sintaxis: Sub Mouseout_objeto()

Parámetros:

Sin parámetros

Nombre: Resizocol

Descripción: Ocurre al cambiar las dimensiones de una columna de un objeto tabla

Sintaxis: Sub Resizocol_objeto(col)

Parámetros:

col: columna a la que se le cambió las dimensiones

Nombre: Resizerow

Descripción: Ocurre al cambiar las dimensiones de una fila de un objeto tabla

Sintaxis: Sub ResizeRow_objeto(fila)

Parámetros:

fila: Fila a la que se le cambió las dimensiones

Eventos del teclado sobre los objetos

Nombre: Keydown

Descripción: Ocurre al presionar una tecla en un objeto

Sintaxis: Sub Keydown_objeto(p1)

Parámetros:

p1: contiene el valor numérico de la tecla que generó el evento

Nombre: Keyup

Descripción: Ocurre al liberar una tecla en un objeto

Sintaxis: Sub Keyup_objeto(p1)

Parámetros:

p1: contiene el valor numérico de la tecla que generó el evento

Nombre: Keyenter

Descripción: Ocurre al presionar la tecla Enter en un objeto

Sintaxis: Sub Keyenter_objeto()

Parámetros:

Sin parámetros

Eventos de los objetos

Nombre: Show

Descripción: Ocurre al mostrarse un objeto

Sintaxis: Sub Show_objeto()

Parámetros:

Sin parámetros

Nombre: Hide

Descripción: Ocurre al ocultarse un objeto

Sintaxis: Sub Hide_objeto()

Parámetros:

Sin parámetros

Nombre: Resize

Descripción: Ocurre al cambiar las dimensiones de un objeto

Sintaxis: Sub Resize_objeto()

Parámetros:

Sin parámetros

Nombre: Rotate

Descripción: Ocurre al rotar un objeto

Sintaxis: Sub Rotate_objeto()

Parámetros:

Sin parámetros

Nombre: Change

Descripción: Ocurre al cambiar las propiedades de un objeto

Sintaxis: Sub Change_objeto()

Parámetros:

Sin parámetros

Nombre: Load

Descripción: Ocurre al crearse un objeto

Sintaxis: Sub Load_objeto()

Parámetros:

Sin parámetros

Nombre: Destroy

Descripción: Ocurre al eliminarse un objeto

Sintaxis: Sub Destroy_objeto()

Parámetros:

Sin parámetros

Nombre: Select

Descripción: Ocurre al seleccionar un elemento en una lista

Sintaxis: Sub Select_objeto(p1)

Parámetros:

p1: contiene el elemento seleccionado en la lista, si la lista es de selección múltiple entonces devuelve una cadena con dichos elementos separados por un caracter de retorno.

Nombre: Linkclick

Descripción: Ocurre al hacer clic en un hipervínculo de un objeto html

Sintaxis: Sub Linkclick_objeto(p1)

Parámetros:

p1: Contiene el valor Href que contiene la página en dicho vínculo.

Nombre: Maxlen

Descripción: Ocurre al alcanzar el número máximo de caracteres en una Caja de Texto

Sintaxis: Sub Maxlen_objeto()

Parámetros:

Sin parámetros

Nombre: Timer

Descripción: Evento periódico del objeto Reloj

Sintaxis: Sub Timer_objeto()

Parámetros:

Sin parámetros

Nombre: Play

Descripción: Ocurre al iniciarse un video en el objeto Video

Sintaxis: Sub Play_objeto()

Parámetros:

Sin parámetros

Nombre: LoadFile

Descripción: Ocurre al cargarse un fichero de video en el objeto Video

Sintaxis: Sub LoadFile_objeto()

Parámetros:

Sin parámetros

Nombre: Stop

Descripción: Ocurre al detenerse por el usuario un video en el objeto Video

Sintaxis: Sub Stop_objeto()

Parámetros:

Sin parámetros

Nombre: Pause

Descripción: Ocurre al poner en pausa un video en el objeto Video

Sintaxis: Sub Pause_objeto()

Parámetros:

Sin parámetros

Nombre: Finished

Descripción: Ocurre al finalizar la reproducción de un video en el objeto Video

Sintaxis: Sub Finished_objeto()

Parámetros:

Sin parámetros

Nombre: ChangeState

Descripción: Ocurre al cambiar el estado de un objeto Video

Sintaxis: Sub ChangeState_objeto()

Parámetros:

Sin parámetros

Eventos de página

Nombre: Pageenter

Descripción: Ocurre al llegar a una página mediante navegación

Sintaxis: Sub Pageenter_pagina(p1)

Parámetros:

p1: Contiene el nombre de la página desde la que se navega.

Nombre: Pageleave

Descripción: Ocurre al dejar una página mediante navegación

Sintaxis: Sub Pageleave_pagina(p1)

Parámetros:

p1: Contiene el nombre de la página a la que se navega.

Nombre: Pagemousedown

Descripción: Ocurre al presionar el botón del mouse sobre una página

Sintaxis: Sub Pagemousedown_pagina(p1)

Parámetros:

p1: contiene el botón del mouse que generó el evento

Nombre: Pagemouseup

Descripción: Ocurre al liberar el botón del mouse sobre una página

Sintaxis: Sub Pagemouseup_pagina(p1)

Parámetros:

p1: contiene el botón del mouse que generó el evento

Nombre: Pagemousedc

Descripción: Ocurre al hacer doble clic sobre una página

Sintaxis: Sub Pagemousedc_pagina(p1)

Parámetros:

p1: contiene el botón del mouse que generó el evento

Nombre: Pagemouseclick

Descripción: Ocurre al hacer clic sobre una página

Sintaxis: Sub Pagemouseclick_pagina(p1)

Parámetros:

p1: contiene el botón del mouse que generó el evento

Nombre: Pagemousedragging

Descripción: Ocurre al arrastrar sobre una página

Sintaxis: Sub Pagemousedragging_pagina(p1,p2)

Parámetros:

p1: contiene la coordenada x del mouse

p2: contiene la coordenada y del mouse

Nombre: Pagemouseover

Descripción: Ocurre al mover el puntero sobre una página

Sintaxis: Sub Pagemouseover_pagina(p1,p2)

Parámetros:

p1: contiene la coordenada x del mouse

p2: contiene la coordenada y del mouse

Nombre: Pagekeydown

Descripción: Ocurre al presionar una tecla en una página

Sintaxis: Sub Pagekeydown_pagina(p1)

Parámetros:

p1: contiene el valor numérico de la tecla que generó el evento

Nombre: Pagekeyup

Descripción: Ocurre al liberar una tecla en una página

Sintaxis: Sub Pagekeyup_pagina(p1)

Parámetros:

p1: contiene el valor numérico de la tecla que generó el evento

Eventos de la aplicación

Estos eventos reemplazan a los eventos de las páginas cuando se emplea el runtime para ejecutar un script (modo script). Por tanto estos no existen en un proyecto con páginas.

Nombre: Appmousedown

Descripción: Ocurre al presionar el botón del mouse sobre la aplicación

Sintaxis: Sub Appmousedown(p1)

Parámetros:

p1: contiene el botón del mouse que generó el evento

Nombre: Appmouseup

Descripción: Ocurre al liberar el botón del mouse sobre la aplicación

Sintaxis: Sub Appmouseup(p1)

Parámetros:

p1: contiene el botón del mouse que generó el evento

Nombre: Appmousedc

Descripción: Ocurre al hacer doble clic sobre la aplicación

Sintaxis: Sub Appmousedc(p1)

Parámetros:

p1: contiene el botón del mouse que generó el evento

Nombre: Appmouseclick

Descripción: Ocurre al hacer clic sobre la aplicación

Sintaxis: Sub Appmouseclick(p1)

Parámetros:

p1: contiene el botón del mouse que generó el evento

Nombre: Appmousedragging

Descripción: Ocurre al arrastrar sobre la aplicación

Sintaxis: Sub Appmousedragging(p1,p2)

Parámetros:

p1: contiene la coordenada x del mouse

p2: contiene la coordenada y del mouse

Nombre: Appmouseover

Descripción: Ocurre al mover el puntero sobre la aplicación

Sintaxis: Sub Appmouseover(p1,p2)

Parámetros:

p1: contiene la coordenada x del mouse

p2: contiene la coordenada y del mouse

Nombre: Appkeydown

Descripción: Ocurre al presionar una tecla en la aplicación

Sintaxis: Sub Appkeydown(p1)

Parámetros:

p1: contiene el valor numérico de la tecla que generó el evento

Nombre: Appkeyup

Descripción: Ocurre al liberar una tecla en la aplicación

Sintaxis: Sub Appkeyup(p1)

Parámetros:

p1: contiene el valor numérico de la tecla que generó el evento

Nombre: AppReady

Descripción: Ocurre cuando después de iniciar, la aplicación está lista.

Sintaxis: Sub AppReady()

Importante: Este evento puede ser empleado tanto en el modo de script como en el de proyecto.

Eventos de la clase ssSonido

Nombre: SPlay

Descripción: Ocurre al iniciarse un sonido en un ssSonido

Sintaxis: Sub SPlay_objeto()

Parámetros:

Sin parámetros

Nombre: SLoadFile

Descripción: Ocurre al cargarse un fichero en el objeto ssSonido

Sintaxis: Sub SLoadFile_objeto()

Parámetros:

Sin parámetros

Nombre: SStop

Descripción: Ocurre al detenerse por el usuario un ssSonido

Sintaxis: Sub SStop_objeto()

Parámetros:

Sin parámetros

Nombre: SPause

Descripción: Ocurre al poner en pausa un ssSonido

Sintaxis: Sub SPause_objeto()

Parámetros:

Sin parámetros

Nombre: SFinished

Descripción: Ocurre al finalizar la reproducción de un ssSonido

Sintaxis: Sub SFinished_objeto()

Parámetros:

Sin parámetros

Nombre: SChangeState

Descripción: Ocurre al cambiar el estado de un ssSonido

Sintaxis: Sub SChangeState_objeto()

Parámetros:

Sin parámetros

A continuación se muestra la lista de eventos que genera cada tipo de objeto:

Imagen:

mousedown
 mouseup
 mousedc
 mouseclick
 mousedragging
 mouseover
 mousein
 mouseout
 show
 hide
 resize
 rotate
 change
 load
 destroy

Botón:

mousedown
 mouseup
 mousedc
 mouseclick
 mousedragging
 mouseover
 mousein
 mouseout
 show
 hide
 resize

rotate
change
load
destroy

Etiqueta

mousedown
mouseup
mousedc
mouseclick
mousedragging
mouseover
mousein
mouseout
show
hide
resize
rotate
change
load
destroy

Polígono

mousedown
mouseup
mousedc
mouseclick
mousedragging
mouseover
mousein
mouseout
show
hide
change
load
destroy

Html

show
hide
resize
change
load
destroy
linkclick
mouseclick
mouseover

Caja texto

mousedown
mousedragging
mouseover
mousein
mouseout
maxlen
show

hide
resize
change
load
destroy
keyenter

Lista

mousedc
show
hide
resize
change
load
destroy
select

Tabla

mousedown
show
hide
resize
change
load
destroy
resizecol
resizerow

Texto

mouseclick
show
hide
resize
change
load
destroy
mousein
mouseout
keydown
keyup

Video

show
hide
resize
change
load
destroy
play
loadfile
stop
finished
changestate
pause

Reloj

timer

Página

pageenter
pageleave
pagemousedown
pagemouseup
pagemousedc
pagemouseclick
pagemouseover
pagemousedragging
pagekeydown
pagekeyup

Aplicación

appmousedown
appmouseup
appmousedc
appmouseclick
appmouseover
appmousedragging
appkeydown
appkeyup

Los objetos básicos del proyector y las clases

Los objetos básicos del proyector (etiquetas, botones, etc.) que vienen incluidos como objetos básicos de Sora Script pueden relacionarse con las clases de wxBasic para crear objetos de usuario más complejos y adaptados a las necesidades del programador. Los nuevos objetos creados sólo serán visibles en tiempo de ejecución puesto que la forma de construirlos está directamente ligada a la construcción de una clase mediante script.

Lo primero que se debe tener presente es que los objetos básicos del proyector (etiquetas, botones, etc) siempre que sean creados en el constructor de una clase o en un procedimiento llamado desde dicho constructor, pasan a pertenecer a dicha clase como miembros de esta. Esto implica que ya no serán accesibles desde fuera del ámbito de dicha clase. Tampoco sus eventos serán capturados fuera sino sólo dentro de la clase a la que pertenezca.

Para que los nuevos objetos creados generen eventos propios emplearemos el método *ssllamada* (ver *ssllamada*).

Ejemplo:

En este ejemplo se muestra la creación de una clase de usuario compuesta por un objeto del proyector (etiqueta). Observe como se

captura el evento `mousedown` de la etiqueta en el interior de la clase para generar un evento nuevo fuera de la clase (`eventoclick`).

Para probar este ejemplo ponga este script en un fichero y ejecútelo mediante el runtime de la siguiente forma:
`runtime.exe fichero.txt`

```
class laetiqueta
    dim minombre
    dim mietiqueta as ssetiqueta

    sub new(nombre,texto,pos as sspunto)
        //almaceno el nombre para usarlo luego en el evento a generar
        minombre=nombre
        //creo el objeto que ya pertenece a esta clase
        mietiqueta=ssetiqueta("eti",texto,pos)
    end sub

    //se captura el evento de la etiqueta creada("eti") y se genera el evento de este objeto
    sub mousedown_etiq(a)
        sllamada(nothing,"eventoclick_" & minombre, a)
    end sub
end class

//probando la clase elaborada
mietiqueta=new laetiqueta("lamia","el texto",sspunto(100,100))

//capturo el evento
sub eventoclick_lamia(boton)
    sMensaje("el boton presionado fue el " & boton)
end sub
```

Hasta aquí todo funcionaría perfectamente, sin embargo este objeto nuevo creado (`laetiqueta`) no podría ser empleada para construir otra clase, puesto que su evento (`eventoclick`) no podría ser capturado sino en el ámbito global. Para evitar esto se emplea el primer parámetro del método `sllamada`. Haciendo un arreglo a esta clase para que pueda ser empleada directamente (como se mostró) o como parte de otra clase, quedaría así:

```
class laetiqueta
    dim minombre
    dim mietiqueta as ssetiqueta
    dim mipadre

    sub new(padre, nombre,texto,pos as sspunto)
        //almaceno el nombre para usarlo luego en el evento a generar
        minombre=nombre
        mipadre=padre
        //creo el objeto que ya pertenece a esta clase
        mietiqueta=ssetiqueta("eti",texto,pos)
    end sub

    //se captura el evento de la etiqueta creada("eti") y se genera el evento de este objeto
    sub mousedown_etiq(a)
        sllamada(mipadre,"eventoclick_" & minombre, a)
    end sub
end class

//probando la clase elaborada
```

```

mietiqueta=new laetiqueta(nothing, "lamia", "el texto", sspunto(100,100))

//capturo el evento
sub eventoclick_lamia(boton)
  ssMensaje("el boton presionado fue el " & boton)
end sub

```

Se han marcado de color verde los cambios para hacerlos más visibles. Observe que ahora se sigue empleando el objeto de igual forma, aunque especificando que no pertenece a ningún padre. Ahora un ejemplo de la utilidad del cambio realizado. Esta modificación permite que esta clase pueda ser utilizada para crear objetos en el ámbito global o dentro de otra clase.

Supongamos que ahora con este nuevo objeto etiqueta se necesita conformar un objeto (por ejemplo textodoble) que emplee más de una etiqueta. En este caso se debe tener presente que los eventos independientes de las etiquetas no se deben recibir en el ámbito global sino dentro de la clase textodoble para luego generar un evento único perteneciente a este nuevo objeto (textodoble). El código podría quedar así:

```

class laetiqueta

  dim minombre
  dim mietiqueta as ssetiqueta

  sub new(nombre,texto,pos as sspunto)
    //almaceno el nombre para usarlo luego en el evento a generar
    minombre=nombre
    //creo el objeto que ya pertenece a esta clase
    mietiqueta=ssetiqueta("eti",texto,pos)
  end sub

  //se captura el evento de la etiqueta creada("eti") y se genera el evento de este objeto
  sub mouseclick_etiq(a)
    ssllamada(nothing,"eventoclick_" & minombre, a)
  end sub

end class

//la clase padre
class textodoble

  dim minombre
  dim unaetiqueta as laetiqueta
  dim otraetiqueta as laetiqueta

  sub new(nombre, pos)
    minombre=nombre
    unaetiqueta=laetiqueta(me,"linea1", "este es el texto 1", pos)
    otraetiqueta=laetiqueta(me,"linea2", "y este es el texto 2", pos)
  end sub

  //capturo el evento de la linea1
  sub eventoclick_linea1(boton)
    ssllamada(nothing,"clicktextodoble_" & minombre, boton)
  end sub

  //capturo el evento de la linea2
  sub eventoclick_linea2(boton)
    ssllamada(nothing,"clicktextodoble_" & minombre, boton)
  end sub

end class

```

```
//probando la clase elaborada
parrafo=textodoble("eltexto",sspunto(100,100))

//capturo el evento de parrafo
sub clicktextodoble_eltexto(boton)
  ssMensaje("el boton presionado fue el " & boton)
end sub
```

Estos elementos sumados a las potencialidades de la herencia de clases permitirán a los programadores crear sus propios objetos con mucha libertad y con grandes posibilidades.

A continuación se muestra la construcción de una clase slider que al ser instanciada permitirá el trabajo con este objeto tan útil que en este caso ha sido creado con dos imágenes.

```
class slider

  dim posx as integer
  dim posy as integer
  dim dx, dy,dxc,dyc
  dim elnombre
  dim valor as integer
  dim cursor as table
  dim base as table

  dim imagenfondo as ssimagen
  dim imagencursor as ssimagen

  sub new(nombre,x,y)
    elnombre=nombre

    valor=0
    posx=x
    posy=y
    //imagen de boton
    cursor = {
      "11 15 37 1",
      "  c #EFEFEF",
      ". c #B9BABB",
      "+ c #8C8D90",
      "@ c #F4F4F6",
      "# c #F1F2F4",
      "$ c #E8E9EF",
      "% c #DADCE1",
      "& c #EFF0F2",
      "* c #B7B9BC",
      "= c #E6E7ED",
      "- c #D8DADF",
      "; c #EDEF1",
      "> c #F5F6F7",
      ", c #E3E4EA",
      "' c #D5D7DC",
      ") c #EBECEf",
      "! c #E1E2E8",
      "~ c #D3D5DA",
      "{ c #E9EAED",
      "] c #DEDFE5",
      "^ c #D1D2D7",
      "/" c #E7E8EC",
      "( c #DCDDE3",
      " c #CED0D5",
      ": c #E5E6EA",
      "< c #DADBE1",
      "[ c #CCCDD2",
      "} c #E3E4E8",
      "| c #D7D8DE",
```



```

~ximagen
~yimagen

end sub

sub mousedragging_boton(x,y)
  if(x>posx-(dx)/2) and (x<posx+(dx)/2) then
    imagencursor.estposicion(sspunto(x,posy))

    long=x-(posx-dx/2)
    valor=long/dx*100

    ssllamada(nothing,"cambio_" & elnombre,abs(valor))
  end if
end sub

function obtenervalor()
  return valor
end function

sub estvalor(nvalor as integer)
  long=dx*nvalor/100
  x=(posx-dx/2)+long
  valor=nvalor
  imagencursor.estposicion("boton",x,posy)
end sub

sub estima(aa)
  imagencursor.estimagenB(aa)
end sub

sub finalize()
~imagencursor
~imagenfondo
end sub

end class

dim eti as ssetiqueta
//creando un slider
a=slider("mislider",100,100)
//creando una etiqueta
eti=ssetiqueta("valor","0",sspunto(100,200))

//capturando el evento del objeto mislider
sub cambio_mislider(v)
  eti.esttexto(v)
end sub

```

El Runtime

Junto con los ficheros de HAEduc se distribuye el runtime que constituye el ejecutable que permite el funcionamiento de todas las aplicaciones elaboradas en este sistema. Generalmente se le encuentra en la misma carpeta del ejecutable con el nombre run.exe para Windows y run para Linux. Independientemente del entorno de desarrollo integrado (IDE) de HAEduc, el runtime puede ser empleado para ejecutar directamente scripts de wxBasic (modo script) con la mayoría de las funciones y objetos que constituyen el lenguaje Sora Script. De lo dicho anteriormente se deduce que es posible incluso

crear un software todo lo complejo que se necesite, escribiendo el script en un fichero y editándolo con cualquier editor de texto plano.

Un ejemplo de script ejecutado directamente sobre el runtime:

```
//genera un mensaje
ssMensaje("Esto es un ejemplo")

//para cerrar la ventana y salir
//ssaplicacionsalir()
```

Este texto lo guardamos en un fichero cualquiera (por ejemplo: codigo.txt). Para ejecutarlo:

Si estamos en Windows bastará arrastrar el fichero codigo.txt y dejarlo caer sobre el runtime (run.exe)

También es posible lograrlo si en una consola escribimos la dirección del runtime seguida de la del código:

```
c:\run.exe c:\codigo.txt
```

De estar en Linux es probable que funcionen ambas variantes aunque en algunos entornos gráficos falla la primera.

De existir algún error en el código, la aplicación se cerrará y se creará un fichero con la información de dicho error de nombre wx.err.

Es importante destacar que los eventos de las páginas ahora no existen, ya que son sustituidos por los eventos de la aplicación de forma tal que por ejemplo el evento Pagemousedown ahora es sustituido por Appmousedown. Esto ocurre porque en modo script no existe la metáfora del libro por lo cual tampoco existen las páginas ni sus eventos.

Por otra parte, también es posible cargar un proyecto de un libro que se estaba editando anteriormente mediante HAEduc (modo de proyecto). Para ello emplearemos el método de consola antes descrito pero ahora agregando los parámetros -p -i antes de la dirección del fichero del libro:

```
c:\run.exe -p -i c:\proyecto\libro.hal
```

Este es similar al método empleado por el IDE de HAEduc para probar los proyectos en modo de diseño.

Palabras reservadas

Las siguientes palabras están reservadas por el lenguaje:

abstract	as	case	class
and	break	catch	close

const	global	nothing	step
constant	if	open	sub
continue	in	option	then
dim	inherits	or	throw
do	input	output	to
each	inv	print	try
else	is	qbasic	until
elseif	line	redo	virtual
elsif	me	return	wend
end	mod	select	while
erase	mybase	self	xor
exit	new	shared	pag
finally	next	shl	
for	noconsole	shr	
function	not	static	

Operadores

A continuación los operadores que pueden ser empleados para crear expresiones, se muestran ordenados en orden de prioridad de operación.

expresión ^ expresión potencia.
 - expresión menos unario.
 expresión * expresión multiplicación
 expresión \ expresión división entera
 expresión / expresión división
 expresión Mod expresión resto de la división entera
 expresión Inv expresión inverso
 expresión + expresión adición
 expresión & expresión concatenación de cadenas
 expresión - expresión sustracción
 expresión = expresión igualdad
 expresión <> expresión desigualdad
 expresión != expresión desigualdad (forma alternativa)
 expresión < expresión menor que
 expresión > expresión mayor que
 expresión <= expresión menor o igual
 expresión >= expresión mayor o igual
 Not expresión negación lógica
 expresión And expresión And lógico
 expresión Or expresión Or lógico
 expresión Xor expresión Xor lógico
 expresión | expresión Or lógico

Operadores de asignación

Variable= expresión

Además de la asignación estandar wxBasic permite otras variantes:

variable += expresión 'variable = variable + expresión
variable -= expresión 'variable = variable - expresión
variable *= expresión 'variable = variable * expresión
variable /= expresión 'variable = variable / expresión
variable &= expresión 'variable = variable & expresión

Funciones nativas del lenguaje

Abs

n2 = Abs(*n1*)

Devuelve el valor absoluto (valor positivo) de *n1*.

'n es 1

n = Abs(1)

ACos

n2 = ACos(*n1*)

Devuelve un ángulo con el coseno igual a *n1*. El argumento debe estar en el rango de -1 a +1

'n es 3.141592654

n = ACos(-1)

AndBits

n3 = AndBits(*n1*, *n2*)

Devuelve el resultado de una operación bitwise and realizada con los valores de *n1* y *n2*.

'n toma el valor 1

n = AndBits(1, 3)

Argv

n2 = Argv(*n1*)

Devuelve el valor del parámetro *n1* pasado a una rutina. Empleese con rutinas que tengan un número indefinido de parámetros.

Function miFuncion(...)

ssMensaje("Se han pasado " & Argc() & " parámetros")

For i = 1 To Argc()

ssMensaje("Parámetro " & i & " es " & Argv(i))

Next

End Function

Asc

n = Asc(*string*)

Devuelve el código ASCII del primer caracter de un string. Las cadenas vacías devuelven un valor cero.

' asigna un 65 a n

n = Asc("A")

ASin

n2 = ASin(*n1*)

Devuelve un ángulo con seno igual a *n1*. El argumento debe estar en el rango de -1 a 1. Un valor entre $-PI/2$ y $+PI/2$ (radianes) será devuelto.

' n es -1.570796327

n = ASin(-1)

ATan

$n2 = \text{ATan}(n1)$

devuelve un ángulo con tangente igual a $n1$. Un valor entre $-\pi/2$ y $\pi/2$ (en radianes) será devuelto.

' n es 0.785398

$n = \text{ATan}(1)$

Chr\$

$string = \text{Chr}\$(n)$

Devuelve un caracter de cadena seguido de un '\n' correspondiente al código ASCII n

' $s = "A"$

$s = \text{Chr}\$(65)$

Close

$\text{Close}(n)$

Close

Cierra un fichero abierto mediante el handler n . si no se especifica el número de fichero, serán cerrados todos los ficheros abiertos.

' pondrá "Hola mundo" en el fichero "temp.txt"

Open "temp.txt" For Output as #1

Print #1, " Hola mundo "

Close #1

Const

$\text{Const nombre} = \text{expresión} [, \dots]$

Crea una constante con valor igual a una expresión. Múltiples constantes pueden ser declaradas en una línea. Las constantes globales no tienen que ser declaradas dentro de los Subs o Funciones con la sentencia *Shared*.

'Crear constante PI

Const PI = 3.14159271

Continue

Continue

Envía al compilador a la línea inicial de un ciclo. Esto sólo es aplicable a los ciclos For y While.

'No procesa números mayores de 5

For i = 1 to 10

If i > 5 Then

Continue

End If

Print i

End For

Cos

$n2 = \text{Cos}(n1)$

Devuelve el coseno de $n1$ en radianes.

' n es 1

$n = \text{Cos}(0)$

Command()

$n = \text{Command}()$

Devuelve la cantidad de parámetros enviados a la aplicación al ejecutarla.

Command(as integer)

val = Command(*p*)

Devuelve el valor de un parámetro (en este caso *p*) enviado a la aplicación al ejecutarla.

Date\$

string = Date\$()

Devuelve una cadena de fecha en el format MM:DD:YYYY. Se puede además emplear Date().

' s contendrá la fecha actual

s = Date\$(0)

Dim

Dim {Shared} *name* []

Dim {Shared} *name* [*maxValue*] [[...]] [, ...]

Dim {Shared} *name* [*minValue* To *maxValue*] [[...]] [, ...]

Dim {Shared} *name* [= *expresión*] [, ...]

Permite crear una variable y reservar memoria.

' Creando un array

Dim a[3 To 10, 5]

' Creando una variable y asignándole un valor

Dim myVar = 100

End

End

Termina la ejecución del programa.

'Salir del programa

End

Erase

Erase *array* []

Erase *array* [...]

Si no se especifican los índices, se elimina el contenido de todo el array. Esto significa que todos sus elementos se inicializan a cero. Para los array dinámicos, esto provoca que se eliminen todos los índices y valores.

Por ejemplo:

'Creando un array estático

Dim a[10]

' Inicializándolo

For i = 1 To 10

a[i] = i

End For

' Inicializar el array

Erase a[]

Si los índices son especificados, sólo estos serán inicializados:

'Creando un array estático

Dim a[10]

' Inicializándolo

For i = 1 To 10

a[i] = i

End For

```
' lo siguiente es como poner a[3] = 0
Erase a[3]
```

Para los array dinámicos:

```
'Creando un array dinámico
Dim a[]
' Inicializa este
For i = 1 To 10
a[i] = i
End For
' Remover el elemento 3
Erase a[3]
```

Eof

```
n = Eof( file handle )
Devuelve true si se ha encontrado un fin de fichero al recorrerlo:
' Leer mientras no encuentre el final del fichero
Open "readme.txt" For Input as #1
While Not Eof( 1 )
Line Input #1, text
ssMensaje(text)
Wend
Close #1
```

fClose

```
fClose( handle )
Cierra el fichero referenciado por handle. (Vea además Close).
' abrir un fichero de texto
handle = fOpen( "myfile.txt", "r" )
' leer hasta el final del fichero
While not Eof( handle ) Do
' leer una línea del fichero e imprimirla
ssMensaje(fGets( handle ))
End While
' cerrar el fichero
fClose( handle )
```

fGets

```
cadena = fGets( handle )
Lee una línea de un fichero de texto y se le asigna a cadena. (Vea además Line Input #)
' abrir un fichero de texto
handle = fOpen( "myfile.txt", "r" )
' leer hasta el final
While not Eof( handle ) Do
' leer una línea de texto e imprimirla
Print fGets( handle )
End While
' cerrar el fichero
fClose( handle )
```

FileExists

```
FileExists( filename )
Devuelve true si un fichero (filename) existe, sino devuelve false. Este no abre el fichero.
' verifica si el fichero existe
If Not FileExists( "myfile.txt" ) then
ssMensaje ("el fichero \"myfile.txt\" no existe.")
```

End If

Fix

$n2 = \text{Fix}(n1)$

Devuelve la porción entera de $n1$. Similar a Floor().

' n es 3

$n = \text{Fix}(3.1415927)$

Floor

$n2 = \text{Floor}(n1)$

Devuelve la porción entera de $n1$. Similar a Fix().

' n es 3

$n = \text{Floor}(3.1415927)$

fOpen

$n = \text{fOpen}(\text{filename}, \text{modo})$

abre un fichero en un *modo* y devuelve la referencia numérica n a este.

Devuelve cero si no se puede abrir el fichero. Los ficheros pueden abrirse en los siguientes modos:

"r" = read

"w" = write

"a" = append

(vea también *Open*)

' abre un fichero de texto en modo de lectura

handle = fOpen("myfile.txt", "r")

' leer hasta el final del fichero

While not Eof(handle) Do

' leer una línea del fichero e imprimirla

ssMensaje(fGets(handle))

End While

' cerrar el fichero

fClose(handle)

fPuts

fPuts(handle, string)

Escribe una cadena en un fichero referenciado mediante *handle*. La cadena será automáticamente adicionada al final del contenido del fichero. (ver además *Print #*).

' abre un fichero en modo de escritura

handle = fOpen("myfile.txt", "w")

' escribe un texto en él

fPuts(handle, "Esta es la primera línea\n")

fPuts(handle, "Esta es la segunda línea\n")

fPuts(handle, "Esta es la tercera línea\n")

' cerrar el fichero

fClose(handle)

Frac

$n2 = \text{Frac}(n1)$

Devuelve la porción no entera de $n1$.

' n es 0.1415927

$n = \text{Frac}(3.1415927)$

FreeFile

n = FreeFile()

Devuelve el próximo valor libre de identificadores de ficheros o cero si no hay ninguno libre.

' abrir el fichero "myfile.txt" en el próximo identificador libre

fileNum = FreeFile()

Open "myfile.txt" For Output As #fileNum

HasKey

n = HasKey(table, key)

Devuelve 1 si existe un identificador (key) de un elemento en una tabla (variable tipo table), de no existir devuelve 0.

Hex\$

string = Hex\$(*n*)

Devuelve la representación Hexadecimal de el número *n*. Sólo de la porción entera de *n* será usada.

' *n* es "c"

n = Hex\$(12)

In

expresión In *array* []

Devuelve true si la expresión es un índice del array dinámico *array* [].

Returns True if *expresión* is a key in the dynamic array *array* []. En un array estático provocará un error.

'verifica si key es un índice de a[]

If key In a[] Then

ssMensaje(key & " es un índice válido en a[]")

End If

Include

Include *filename*

Inserta un fichero en el código fuente. Esta instrucción no puede estar en el interior de una estructura.

'Include the file "defs.inc"

Include "defs.inc"

Indexes

count = Indexes(*array* [])

Devuelve el número de índices de un *array*.

' muestra información sobre los índices de un array

count = Indexes(a[])

ssMensaje ("Hay " & count & " índices en este array")

For i = 1 To count

bottom = LBound(a[], i)

top = UBound(a[], i)

ssMensaje ("Índice " & i & " está entre " & bottom & " y " & top)

Next

Instr

n = Instr(*cadena*, *subcadena* [*pos*])

Devuelve la posición de una subcadena en una cadena. Si no se encuentra devuelve cero. Un valor opcional de *pos* establece la posición desde donde comenzará la búsqueda.

```
' le asigna 2 a n
n = Instr( "wxBasic", "Basic" )
```

Int

```
n2 = Int( n1 )
Devuelve la porción entera de n1.
' asigna 12 a n
n = Int( 12.44 )
```

LBound

```
n = LBound( array[], indice )
Devuelve el menor valor del rango de valores de un índice (dimensión) en un array.
' cantidad de índices (dimensiones) de un array
count = Indexes( a[] )
ssMensaje ("Este array contiene " & count & " índices")
For i = 1 To count
' menor valor del rango para esta dimensión
bottom = LBound( a[], i )
' mayor valor del rango para esta dimensión
top = UBound( a[], i )
ssMensaje("el índice " & i & " va desde " & bottom & " hasta " & top
Next
```

LCase\$

```
string2 = LCase$( string1 )
Devuelve una cadena en minúsculas.
' s tomara el valor "wxBasic"
s = LCase$("wxBasic")
```

Left\$

```
string2 = Left$( string1, n )
Devuelve una cadena formada por n elementos desde la izquierda de la
cadena string1.
' s tomará el valor "wx"
s = Left$( "wxBasic", 2 )
```

Len

Length

```
n = Len( string )
n = Length( string )
Devuelve la longitud de una cadena.
' n tomará el valor 7
n = Len("wxBasic")
```

Line Input

```
Line Input # file handle, string
Asigna una cadena a la próxima línea de texto en el fichero handle.
' asigna las primeras 10 líneas del fichero "readme.txt" a los respectivos elementos de un array
Dim text[10]
Open "readme.txt" For Input As #1
For i = 1 To 10
Line Input #1, text[i]
```

Next
Close #1

Loc

position = Loc(*file handle*)

Devuelve la posición en la que actualmente se está leyendo en un fichero. Solo funciona si el fichero ha sido abierto en modo *Input*. Este funciona similar a Seek().

' obtiene posición actual en el fichero
ssMensaje("la posición actual de lectura es " & Loc(1))

Lof

n = Lof(*handle*)

Devuelve la longitud de un fichero. . Solo funciona si el fichero ha sido abierto en modo *Input*.

' Obtiene la longitud del fichero *myfile.txt*
filename = "myfile.txt"
Open filename For Input As #1
ssMensaje("El fichero " & filename & " tiene " & Lof(1) & " bytes de tamaño"
Close(1)

Log

n2 = Log(*n1*)

Devuelve el logaritmo natural de *n1*.

' n tundra el valor 2.302585
n = Log(10)

LTrim\$

string2 = LTrim(*string1*)

devuelve a *string1* sin espacios en blanco ni tabs a la izquierda.

' s tundra valor "wxBasic"
s = LTrim\$(" wxBasic")

Mid\$

string2 = Mid\$(*string1*, *pos*, *n*)

Devuelve una subcadena de *string1* formada por los *n* caracteres desde *pos*.

' s = "Ba"
s = Mid\$("wxBasic", 3, 2)

New

objeto = New *class*(listadeargumentos)

crea una instancia de un objeto. Estos objetos deben ser destruidos con *Delete*.

Para más detalles consulte la referencia a este en páginas anteriores.

NotBits

n2 = NotBits(*n1*)

Devuelve el resultado de la operación bitwise not de *n1*.

' n toma el valor -2
n = AndBits(1)

Open

Open *filename* For Input | Output | Append As #*filenumber*

Abre un fichero para lectura o escritura. Con Input se abre un fichero para lectura, con Output para escritura y con Append se abre un fichero existente para escribir al final del mismo.

```
' imprime 10 números en el fichero "temp.txt"
Open "temp.txt" For Output As #1
For i = 1 to 10
Print #1, i
Next
Close #1
```

Option Explicit

Option Explicit

Normalmente wxBasic crea variables automáticamente al asignarle un valor a alguna previamente no creada. Con Option Explicit establece la declaración obligatoria de variables mediante el Dim o Shared para el caso de los procedimientos y funciones. Este debe ponerse como primera declaración en el fichero para entonces afectar a las siguientes declaraciones.

```
' hace que todas las variables sean declaradas explícitamente.
Option Explicit
```

OrBits

$n3 = \text{OrBits}(n1, n2)$

Devuelve el resultado de la operación bitwise or de $n1$ y $n2$.

```
' n toma el valor 3
n = OrBits( 1, 3 )
```

Print

Print # *handle*, *expresión* [,][;][*expresión*]

Evalúa e imprime una expresión en un fichero.

```
' imprime "Hello, wxBasic" en el fichero "temp"
Open "temp" For Output as #1
Print #1, "Hello, wxBasic"
Close #1
```

Randomize

Randomize([n])

Establece un valor inicial del generador de números aleatorios. Si no se establece el valor n este tomará el valor actual de Timer.

```
'Establece el valor inicial del generador al del Timer
Randomize( Timer )
```

ReadByte

$n = \text{ReadByte}(\text{file handle})$

Devuelve el byte de la posición actual en un fichero abierto mediante el *handle*.

```
' muestra el fichero "myfile.txt" byte por byte
Open "myfile.txt" For Input As #1
While Not Eof( 1 )
Seek( 1 )
ssMensaje(Chr$(ReadByte(1)))
End While
Close(1)
```

Reverse\$

```
string2 = Reverse$( string1 )
```

Devuelve una cadena invertida

```
' asigna a s el valor "4321"
```

```
s = Reverse("1234")
```

Right\$

```
string2 = Right$( string1, n )
```

Devuelve los n caracteres situados a la derecha de *string1*.

```
' Set s to "Basic"
```

```
s = Right$( "wxBasic", 5 )
```

RInstr

```
n = RInstr( string, substring )
```

Devuelve la posición de un substring en un string, haciendo una búsqueda desde la derecha.

```
' asigna a n el valor 3
```

```
n = RInstr( "wxBasic", "Basic" )
```

Rnd

```
n2 = Rnd( n1 )
```

Devuelve un número aleatorio entre 1 y n.

```
' le asigna a n un valor aleatorio entre 1 y 10
```

```
n = Rnd( 10 )
```

Round

```
n2 = Round( n1 )
```

Devuelve n1 redondeado.

```
' le asigna a n un 12
```

```
n = Round( 12.3 )
```

RTrim\$

```
string2 = RTrim$( string1 )
```

Devuelve *string1* sin espacios en blanco ni tabs en su extreme derecho.

```
' Le asigna a s el valor "wxBasic"
```

```
s = RTrim$( "wxBasic " )
```

Seek

```
posicion = Seek( handle )
```

```
sucedio = Seek( handle, posicion )
```

Devuelve la posición actual en un fichero. Si el argumento opcional es incluido, se mueve hasta esta posición en el fichero, devolviendo true si sucede. Vea además Loc().

```
' imprime "myfile.txt" byte por byte
```

```
Open "myfile.txt" For Input As #1
```

```
While Not Eof( 1 )
```

```
Seek( 1 )
```

```
ssMensaje(Chr(GetByte(1)))
```

```
End While
```

```
Close(1)
```

Select ... End Select

Select Case *expresión*

```
Case tests
commands
[Case tests
commands ]
[Case Else
commands]
End Select
```

Ejecuta código opcionalmente, dependiendo del valor de una expresión. Los valores, si son más de uno, pueden estar separados por comas, y pueden tener las siguientes formas:

valormenor a valormayor

```
Is = expresion
Is <> expresion
Is > expresion
Is < expresion
Is <= expresion
Is >= expresión
```

```
' muestra el nombre de un número
Select Case n
Case 1
ssMensaje("Uno")
Case 2
ssMensaje("Dos")
Case 3
ssMensaje("Tres")
Case 4, 5, 6
ssMensaje("Cuatro, Cinco o Seis")
Case Is < 10
ssMensaje("Menor que 10")
Case Else
ssMensaje("Muy grande")
End Select
```

Sgn

$n2 = \text{Sgn}(n1)$

Devuelve 1 si n1 es positivo, -1 si este es negativo, y 0 si este es cero.

```
' Imprime el signo de n
Select Case Sgn( n )
Case 1
ssMensaje("El número es positive")
Case -1
ssMensaje("El número es negativo")
Case 0
ssMensaje("El número es cero")
End Case
```

Shared

Shared *variable* [, *variable*...]

Usado para acceder a variables declaradas a nivel de módulo.

```
' Declara una variable a nivel de módulo
Dim counter = 0
Sub incrementCounter()
' hacer referencia a la variable global
Shared counter
' Incrementar el contador
counter = counter + 1
End Sub
```

Shell

Shell(*command string*)

Envía cadenas de comandos a la consola para ser ejecutados. Los comandos dependen del sistema operativo.

' En DOS, elimina todos los ficheros terminados en .TMP

Shell("del *.tmp")

Sin(n)

$n2 = \text{Sin}(n1)$

Devuelve el seno de $n1$.

' asigna a n el valor 0.841471

$n = \text{Sin}(1)$

Space\$

$string2 = \text{Space}\$(string1)$

Devuelve una cadena construida con n espacios.

' asigna a s la cadena " " (tres espacios)

$s = \text{Space}\$(3)$

Sqr

$n2 = \text{Sqr}(n1)$

Devuelve la raíz cuadrada de $n1$.

' asigna a n el valor 3

$n = \text{Sqr}(9)$

Str\$

$string = \text{Str}\$(n)$

Devuelve a n como una cadena de caracteres.

' Asigna a s la cadena "123"

$s = \text{Str}\$(123)$

String\$

$string = \text{String}\$(n, char)$

Construye una cadena con n caracteres *char*.

' Asigna a s la cadena "---"

$s = \text{String}\$(3, "-")$

Tan

$n2 = \text{Tan}(n1)$

Devuelve la tangente del ángulo $n1$.

' Asigna a n el valor 0.557408

$n = \text{Tan}(0)$

Ticks

$n = \text{Ticks}()$

Devuelve el número de ticks desde que el sistema operativo inició. Esto depende del sistema operativo.

' asigna a n el número de ticks que demora el ciclo.

$startTicks = \text{Ticks}()$

For i = 1 To 1000

Next

$n = \text{Ticks}() - startTicks$

Time\$

string = Time\$()

Devuelve la hora actual del sistema con el formato HH:MM:SS.

' Establece a s la hora actual

s = Time\$()

TypeOf\$

string = TypeOf\$(*expresión*)

Devuelve el nombre del tipo de datos que devuelve una expresión. Los tipos son: "number", "string", "object" and "unknown".

' Asigna a s la cadena "string"

s = TypeOf\$("wxBasic")

UBound

number = UBound(*array*[], *index*)

Devuelve el mayor valor del rango de valores de un índice (dimensión) en un array.

' cantidad de índices (dimensiones) de un array

count = Indexes(a[])

ssMensaje("Este array contiene " & count & " índices")

For i = 1 To count

' menor valor del rango para esta dimensión

bottom = LBound(a[], i)

' mayor valor del rango para esta dimensión

top = UBound(a[], i)

ssMensaje("el índice " & i & " va desde " & bottom & " hasta " & top

Next

UCase\$

string2 = UCase\$(*string1*)

Devuelve una cadena convertida a minúscula.

' Asigna a s la cadena "WXBASIC"

s = UCase\$("wxBasic")

Val

n = Val(*string*)

Devuelve el valor numérico contenido en una cadena. Si la cadena esta vacía, devuelve cero.

' asigna a n el valor 12

n = Val("12")

While ... Wend

While ... End While

While *expresión*

commands

[Exit While]

[Continue]

Wend | End While

Ejecuta comandos mientras la expresión se mantenga siendo true.

'contar desde 1 hasta 10

i = 1

While i <= 10

ssMensaje(i)

i = i + 1

Wend

WriteByte

WriteByte(*file handle*, *n*)

Escribe el byte *n* en la posición actual en el fichero *handle*. Esta es usada fundamentalmente para la escritura binaria en ficheros.

```
' copia binaria de un fichero
Open "myfile.txt" For Input As #1
Open "newfile.txt" For Output As #2
While Not Eof( 1 )
WriteByte( 1, ReadByte(2))
End While
Close
```

XorBits

n3 = XorBits(*n1*, *n2*)

Devuelve el bitwise *xor* entre *n1* y *n2*.

' n toma el valor 2

n = XorBits(1, 3)