ESB Routines Library {button &Top/Units,JI(`',`IDH_Library_ESB_Routines')} {button &Types,JI(`',`IDH_Library_ESB_Routines_OtherTypes')} {button &Routines,JI(`',`IDH_Library_ESB_Routines_Routines')}

Units
ESBRtns

Other Types
String16
String32
TBitList
TLongBitList

Routines
BitIsSet
Bits2Str
BitsSet
BlankStr
Boolean2Char
Boolean2TF
Boolean2YN
Booleans2BitList
CentreChStr
CentreStr
ClearAllBits
ClearAllLBits
ClearBit
ClearLBit
Comp2CEStr
Comp2CStr
Comp2EStr
Comp2Str
DashStr
DDashStr
Double2EStr
Double2Str
ESBClear
ESBMoveOfs
ESBSet
Ext2CEStr
Ext2EStr
Ext2EStr2
Ext2Str
FillStr
FlipAllBits
FlipAllLBits
FlipBit
FlipLBit
HashStr
LBitIsSet
LBits2Str
LeftAlignStr
LeftStr
LeftTillStr

LInt2CEStr
LInt2CStr
LInt2EStr
LInt2Str
LInt2ZBEStr
LInt2ZBStr
LInt2ZStr
PadChLeftStr
PadChRightStr
PadLeftStr
PadRightStr
ReplaceChStr
ReverseBits
RightAfterChStr
RightAfterStr
RightAlignStr
RightStr
SetAllBits
SetAllLBits
SetBit
SetLBit
Single2EStr
Single2Str
StarStr
Str2Bits
Str2Byte
Str2Ext
Str2Int
Str2LInt
Str2SInt
Str2SmallInt
Str2Word
StripChStr
StripLChStr
StripTChStr
TF2Boolean
YN2Boolean

Legend

 - Marks that the item has an associated example. (this bitmap is a hyperlink.)
 - Marks that the item has documented bugs.
 - Marks that the item has documented todo's.
(A todo is something which should be fixed before the next release (or "real soon"!))

Relevant to classes and interfaces only

 - Marks that the class/interface has a property, method or event with examples.
 - Marks that the class/interface has a property, method or event with documented bugs.
 - Marks that the class/interface has a property, method or event with documented todo's.

Note that a symbol in the last group is not present if the corresponding symbol in the first group is present.

Uses
(bold - interface)

SysUtils

Used by
(bold - interface)

ESBRtns Unit {button &Top,JI(`',`IDH_Unit_ESBRtns')} {button &Types,JI(`',`IDH_UnitTopic_ESBRtns_OtherTypes')} {button &Routines,JI(`',`IDH_UnitTopic_ESBRtns_Routines')} {button &Const,JI(`',`IDH_UnitTopic_ESBRtns_GlobalConstants')}

Dependencies Legend

Description

ESB Routines Collection v1.2 Miscellaneous Routines to enhance your 32-bit Delphi Programming including:

- 16-bit Bit Lists
- Block Operations
- various String Routines and Conversions

These routines are used by ESB Consultancy within the development of their Customised Application.

We do ask that if this code helps you in you development that you send as an email mailto:esb@gold.net.au or even a local postcard. It would also be nice if you gave us a mention in your About Box or Help File.

ESB Consultancy Home Page: http://www.gold.net.au/~esb

Mail Address: PO Box 2259, Boulder, WA 6432 AUSTRALIA

History:

v1.2 2 Sep 1998

- Delphi 4 Support Added
- Comments changed to be Time2Help compliant
- Help File Added
- Max/Min of Data types removed as High/Low should be used
- Byte2Str removed as LInt2Str can be used in it's place
- Added String To Integer conversions for Smallint, LongWord (4) and Int64(D4)

v1.1 14 Nov 1997

- 32-Bit Bit Lists added

v1.0 Intial Release

Other Types

String16
String32
TBitList
  Used for a Bit List of 16 bits from 15 -> 0
TLongBitList
  Used for a Bit List of 32 bits from 31 -> 0

Routines

BitIsSet
Returns True if Specified Bit in the BitList is 1
Bits2Str
Converts a Bit list to a string of '1' and '0'.
BitsSet
Returns a number from 0 -> 16 indicating the number of Bits Set
BlankStr
Returns a string composed of N blank spaces (i.e.
Boolean2Char
Converts a Boolean Value into the corresponding Character:
True -> TrueChar
False -> FalseChar

Boolean2TF
Converts a Boolean Value into the corresponding Character:
True -> 'T'
False -> 'F'

Boolean2YN
Converts a Boolean Value into the corresponding Character:
True -> 'Y'
False -> 'N'

Booleans2BitList
Converts an Array of Boolean into a BitList.
CentreChStr
Returns a string with specified characters added to the beginning and end of the string to in effect centre the string within the given length.
CentreStr
Returns a string with blank spaces added to the beginning and end of the string to in effect centre the string within the given length.
ClearAllBits
Sets all Bits in the BitList to 0
ClearAllLBits
Sets all Bits in a LongBitList to 0
ClearBit
Sets specified Bit in the BitList to 0
ClearLBit
Comp2CEStr
Converts a Comp (Integral) Real into a Comma'ed String without Padding
Comp2CStr
Converts a Comp (Integral) Real into a Comma'ed String of specified Length, Len, NumPadCh used for Left padding
Comp2EStr
Converts a Comp (Integral) Real into an exact String, No padding
Comp2Str
Converts a Comp (Integral) Real into a String of specified Length, using NumPadCh for Left Padding
DashStr
Returns a string composed of N occurrences of '-'.
DDashStr
Returns a string composed of N occurrences of '='.
Double2EStr
Converts a Double Real into an exact String, No padding, with given number of Decimal Places

Double2Str
Converts a Double Real into a String of specified Length, using NumPadCh for Left Padding, and with Specified number of Decimals

ESBClear
Fills given structure with specified number of 0 values, effectively clearing it.

ESBMoveOfs
Moves Size bytes from Source starting at Ofs1 to destination starting at Ofs 2 using fast dword moves.

ESBSet
Fills given structure with specified number of $FF values, effectively setting it.

Ext2CEStr
Converts an Extended Real into an exact String, No padding, with given number of Decimal Places, with Commas separating thousands

Ext2EStr
Converts an Extended Real into an exact String, No padding, with given number of Decimal Places

Ext2EStr2
Converts an Extended Real into an exact String, No padding, with at most the specified number of Decimal Places

Ext2Str
Converts an Extended Real into a String of specified Length, using NumPadCh for Left Padding, and with Specified number of Decimals

FillStr
Returns a string composed of N occurrences of Ch.

FlipAllBits
Flips all Bits in the BitList, i.e 1 -> 0 and 0 -> 1

FlipAllLBits
Flips all Bits in a LongBitList, i.e 1 -> 0 and 0 -> 1

FlipBit
Flips specified Bit in the BitList, i.e.

FlipLBit
Flips specified Bit in a LongBitList, i.e.

HashStr
Returns a string composed of N occurrences of '#'.

LBitIsSet
Returns True if Specified Bit in a LongBitList is 1

LBits2Str
Converts a Long Bit list to a string of '1' and '0'.

LeftAlignStr
Returns a string of Length N with blank spaces added to the **end** of the string if S is too short, or returning the N Left-most characters of S if S is too long.

LeftStr
Returns the substring consisting of the first N characters of S.

LeftTillStr
Returns the substring consisting of the characters from S up to but not including the specified one.

LInt2CEStr
Convert a LongInt into a Comma'ed String without Padding

LInt2CStr
Convert a LongInt into a Comma'ed String of length Len, with NumPadCh Padding to the Left

LInt2EStr
Convert a LongInt into an exact String, No Padding

LInt2Str
Converts a LongInt into a String of length N withNumPadCh Padding to the Left

LInt2ZBEStr
Convert a LongInt into an exact String, No Padding, with null returned if Value is 0
LInt2ZBStr
Converts a LongInt into a String of length N withNumPadCh Padding to the Left, with blanks returned if Value is 0
LInt2ZStr
Converts a LongInt into a String of length N withNumPadCh Padding to the Left
PadChLeftStr
Returns a string with specified characters added to the beginning of the string until the string is of the given length.
PadChRightStr
Returns a string with specified characters added to the end of the string until the string is of the given length.
PadLeftStr
Returns a string with blank spaces added to the beginning of the string until the string is of the given length.
PadRightStr
Returns a string with blank spaces added to the end of the string until the string is of the given length.
ReplaceChStr
Returns the String with all occurrences of OldCh character replaced with NewCh character.
ReverseBits
Reverses the Bit List, i.e.
RightAfterChStr
Returns the sub-string to the right AFTER the first ocurrence of specifiec character.
RightAfterStr
Returns the sub-string to the right AFTER the first N Characters.
RightAlignStr
Returns a string of Length N with blank spaces added to the **beginning** of the string if S is too short, or returning the N Left-most characters of S if S is too long.
RightStr
Returns the substring consisting of the last N characters of S.
SetAllBits
Sets all Bits in the BitList to 1
SetAllLBits
Sets all Bits in a LongBitList to 1
SetBit
Sets specified Bit in the BitList to 1
SetLBit
Sets specified Bit in a LongBitList to 1
Single2EStr
Converts a Single Real into an exact String, No padding, with given number of Decimal Places
Single2Str
Converts an Single Real into a String of specified Length, using NumPadCh for Left Padding, and with Specified number of Decimals
StarStr
Returns a string composed of N occurrences of '*'.
Str2Bits
Converts a string of '1' and '0' into a BitList.
Str2Byte
Converts a String into a Byte
Str2Ext
Converts a String into an Extended Real
Str2Int

Converts a String into an Integer

Str2LInt
Converts a String into a LongInt

Str2SInt
Converts a String into a ShortInt

Str2SmallInt
Converts a String into a SmallInt

Str2Word
Converts a String into a Word

StripChStr
Returns the String with all specified leading and trailing characters removed.

StripLChStr
Returns the String with all specified leading characters removed.

StripTChStr
Returns the String with all specified trailing characters removed.

TF2Boolean
Converts a Character Value into its corresponding Boolean value:
'T', 't' -> True
Otherwise -> False

YN2Boolean
Converts a Character Value into its corresponding Boolean value:
'Y', 'y' -> True
Otherwise -> False

Global Constants

NumPadCh
Character to use for Left Hand Padding of Numerics

BitIsSet routine

Unit
ESBRtns

Declaration
```
Function BitIsSet(const Body: TBitList; const I: Byte): Boolean;
```

Description
Returns True if Specified Bit in the BitList is 1

Implementation

```
function BitIsSet (const Body: TBitList; const I: Byte): Boolean;

begin
   Result := (Body and ($0001 shl I)) <> 0
End;
```

Bits2Str routine

Unit
ESBRtns

Declaration
```
Function Bits2Str(const Body: TBitList): String16;
```

Description
Converts a Bit list to a string of '1' and '0'.

Implementation

```
function Bits2Str (const Body: TBitList): String16;
var
   I: Integer;
begin
   SetLength (Result, 16);
   for I := 0 to 15 do
      if BitIsSet (Body, I) then
         Result [I + 1] := '1'
      else
         Result [I + 1] := '0';
End;
```

BitsSet routine

Unit
ESBRtns

Declaration
```
Function BitsSet(const Body: TBitList): Byte;
```

Description
Returns a number from 0 -> 16 indicating the number of Bits Set

Implementation

```
function BitsSet (const Body: TBitList): Byte; assembler;
asm
      mov  dx, ax       // Place BitList into BX
      xor   ax, ax       // Clear AX
      mov  cx, 16       // Move 16 into CX
   @2:   shl  dx, 1       // Shift Left
      jnc   @1          // if no carry then no increment
      inc   ax
   @1:   loop @2
End;
```

BlankStr routine

Unit
ESBRtns

Declaration
```
Function BlankStr(const N : Integer): string;
```

Description
Returns a string composed of N blank spaces (i.e. #32)

Implementation

```
function BlankStr (const N : Integer): string;
begin
   Result := FillStr (' ', N);
End;
```

Boolean2Char routine

Unit
ESBRtns

Declaration
```
Function Boolean2Char(const B : Boolean;   TrueChar, FalseChar: Char): Char;
```

Description
Converts a Boolean Value into the corresponding Character:

True -> TrueChar

False -> FalseChar

Implementation

```
function Boolean2Char (const B : Boolean;
   TrueChar, FalseChar: Char): Char;
begin
   if B then
      Result := TrueChar
   else
      Result := FalseChar;
End;
```

Boolean2TF routine

Unit
ESBRtns

Declaration
```
Function Boolean2TF(const B : Boolean): Char;
```

Description
Converts a Boolean Value into the corresponding Character:

True -> 'T'

False -> 'F'

Implementation

```
function Boolean2TF (const B : Boolean): Char;
begin
   if B then
      Result := 'T'
   else
      Result := 'F';
End;
```

Boolean2YN routine

Unit
ESBRtns

Declaration
```
Function Boolean2YN(const B : Boolean): Char;
```

Description
Converts a Boolean Value into the corresponding Character:

True -> 'Y'

False -> 'N'

Implementation

```
function Boolean2YN (const B : Boolean): Char;
begin
   if B then
      Result := 'Y'
   else
      Result := 'N';
End;
```

Booleans2BitList routine

Unit
ESBRtns

Declaration
```
Function Booleans2BitList(const B: array of Boolean): TBitList;
```

Description
Converts an Array of Boolean into a BitList. Only the first 16 Booleans will be used

Implementation

```
function Booleans2BitList (const B: array of Boolean): TBitList;
var
   I: Integer;
begin
   Result := 0;
   for I := 0 to High (B) do
      if B [I] then
         SetBit (Result, 0);
End;
```

CentreChStr routine

Unit
ESBRtns

Declaration
```
Function CentreChStr(const S : String; const Ch : Char;   const Len :
Integer): String;
```

Description
Returns a string with specified characters added to the beginning and end of the string to in effect centre the string within the given length.

If Length (S) >= Len then NO padding occurs, and S is returned.

Implementation

```
function CentreChStr (const S : String; const Ch : Char;
   const Len : Integer): String;
var
   N, M: Integer;
begin
   N := Length (S);
   if N < Len then
   begin
      M := Len - N;
      if Odd (M) then
         Result := FillStr (Ch, M div 2) + S
            + FillStr (Ch, M div 2 + 1)
      else
         Result := FillStr (Ch, M div 2) + S
            + FillStr (Ch, M div 2);
   end
   else
      Result := S;
End;
```

CentreStr routine

Unit
ESBRtns

Declaration
```
Function CentreStr(const S : String; const Len : Integer): String;
```

Description
Returns a string with blank spaces added to the beginning and end of the string to in effect centre the string within the given length.

If Length (S) >= Len then NO padding occurs, and S is returned.

Implementation

```
function CentreStr (const S : String; const Len : Integer): String;
var
   N, M: Integer;
begin
   N := Length (S);
   if N < Len then
   begin
      M := Len - N;
      if Odd (M) then
         Result := BlankStr (M div 2) + S
            + BlankStr (M div 2 + 1)
      else
         Result := BlankStr (M div 2) + S
            + BlankStr (M div 2);
   end
   else
      Result := S;
End;
```

ClearAllBits routine

Unit
ESBRtns

Declaration
```
Procedure ClearAllBits(var Body: TBitList);
```

Description
Sets all Bits in the BitList to 0

Implementation

```
procedure ClearAllBits (var Body: TBitList);

begin
   Body:= $0000
End;
```

ClearAllLBits routine

Unit
ESBRtns

Declaration
```
Procedure ClearAllLBits(var Body: TLongBitList);
```

Description
Sets all Bits in a LongBitList to 0

Implementation

```
procedure ClearAllLBits (var Body: TLongBitList);
begin
   Body:= $00000000
End;
```

ClearBit routine

Unit
ESBRtns

Declaration
```
Procedure ClearBit(var Body: TBitList; const I: Byte);
```

Description
Sets specified Bit in the BitList to 0

Implementation

```
procedure ClearBit (var Body: TBitList; const I: Byte);

begin
   Body:= Body and (not ($0001 shl I))
End;
```

ClearLBit routine

Unit
ESBRtns

Declaration
```
Procedure ClearLBit(var Body: TLongBitList; const I: Byte);
```

Implementation
```
procedure ClearLBit (var Body: TLongBitList; const I: Byte);
begin
   Body:= Body and (not (TLongBitList (1) shl I))
End;
```

Comp2CEStr routine

Unit
ESBRtns

Declaration
```
Function Comp2CEStr(const C : Comp): string;
```

Description
Converts a Comp (Integral) Real into a Comma'ed String without Padding

Implementation

```
function Comp2CEStr (const C : Comp): string;
var
   LS, L, I : Integer;
   Temp : string;
begin
   Result := Comp2EStr (C);
   LS := Length (Result);
   L := (LS - 1) div 3;
   Temp := '';
   for I := 1 to L do
      Temp :=  ThousandSeparator + Copy (Result, LS - 3 * I + 1, 3) + Temp;
   Result := Copy (Result, 1, (LS - 1) mod 3 + 1) + Temp;
End;
```

Comp2CStr routine

Unit
ESBRtns

Declaration
```
Function Comp2CStr(const C : Comp; const Len : Byte): string;
```

Description
Converts a Comp (Integral) Real into a Comma'ed String of specified Length, Len, NumPadCh used for Left padding

Implementation

```
function Comp2CStr (const C : Comp; const Len : Byte): string;
begin
   Result := Comp2CEStr (C);
   Result := PadChLeftStr (LeftStr (Result, Len), NumPadCh, Len);
End;
```

Comp2EStr routine

Unit
ESBRtns

Declaration
```
Function Comp2EStr(const C: Comp): String;
```

Description
Converts a Comp (Integral) Real into an exact String, No padding

Implementation

```
function Comp2EStr (const C: Comp): String;
begin
   try
      Result := FloatToStrF (C, ffFixed, 18, 0)
   except
      Result := '';
   end;
End;
```

Comp2Str routine

Unit
ESBRtns

Declaration
```
Function Comp2Str(const C: Comp; const Len : Byte): String;
```

Description
Converts a Comp (Integral) Real into a String of specified Length, using NumPadCh for Left Padding

Implementation

```
function Comp2Str (const C: Comp; const Len: Byte): String;
begin
   try
      Result := FloatToStrF (C, ffFixed, 18, 0)
   except
      Result := '';
   end;
   Result := PadChLeftStr (LeftStr (Result, Len), NumPadCh, Len);
End;
```

DashStr routine

Unit
ESBRtns

Declaration
```
Function DashStr(const N : Integer): String;
```

Description
Returns a string composed of N occurrences of '-'.

Implementation

```
function DashStr (const N : Integer): string;
begin
   Result := FillStr ('-', N);
End;
```

DDashStr routine

Unit
ESBRtns

Declaration
```
Function DDashStr(const N : Integer): string;
```

Description
Returns a string composed of N occurrences of '='.

Implementation

```
function DDashStr (const N : Integer): string;
begin
   Result := FillStr ('=', N);
End;
```

Double2EStr routine

Unit
ESBRtns

Declaration
```
Function Double2EStr(const D: Double; const Decimals: Byte): String;
```

Description
Converts a Double Real into an exact String, No padding, with given number of Decimal Places

Implementation

```
function Double2EStr (const D: Double; const Decimals: Byte): String;
begin
   try
      Result := FloatToStrF (D, ffFixed, 15, Decimals)
   except
      Result := '';
   end;
End;
```

Double2Str routine

Unit
ESBRtns

Declaration
```
Function Double2Str(const D: Double; const Len, Decimals: Byte): String;
```

Description
Converts a Double Real into a String of specified Length, using NumPadCh for Left Padding, and with Specified number of Decimals

Implementation

```
function Double2Str (const D: Double; const Len, Decimals: Byte): String;
begin
   try
      Result := FloatToStrF (D, ffFixed, 15, Decimals)
   except
      Result := '';
   end;
   Result := PadChLeftStr (LeftStr (Result, Len), NumPadCh, Len);
End;
```

ESBClear routine

Unit
ESBRtns

Declaration
```
Procedure ESBClear(var Dest; const Size: Integer);
```

Description
Fills given structure with specified number of 0 values, effectively clearing it.

Implementation

```
procedure ESBClear (var Dest; const Size: Integer);
begin
   FillChar (Dest, Size, $00);
End;
```

ESBMoveOfs routine

Unit
ESBRtns

Declaration
```
Procedure ESBMoveOfs(const Source; const Ofs1: Integer;   var Dest; const
Ofs2: Integer; const Size: Integer);
```

Description
Moves Size bytes from Source starting at Ofs1 to destination starting at Ofs 2 using fast dword moves. BASM

Implementation

```
procedure ESBMoveOfs (const Source; const Ofs1: Integer;
   var Dest; const Ofs2: Integer; const Size: Integer);
asm
      push    esi
      push    edi

      mov     esi, Source
      add    esi, Ofs1
      mov     edi, Dest
      add    edi, Ofs2

      mov     eax, Size
      mov     ecx, eax

      cmp     edi,esi
      jg      @@DOWN
      je      @@EXIT

      sar     ecx,2           //copy count DIV 4 dwords
      js      @@EXIT

      rep     movsd

      mov     ecx,eax
      and     ecx,03h
      rep     movsb           //copy count MOD 4 bytes
      jmp     @@EXIT

@@DOWN:
      lea     esi,[esi+ecx-4] // point ESI to last dword of source
      lea     edi,[edi+ecx-4] // point EDI to last dword of dest

      sar     ecx,2             // copy count DIV 4 dwords
      js      @@EXIT
      std
      rep     movsd

      mov     ecx,eax
```

```
      and      ecx,03h          // Copy count MOD 4 bytes
      add      esi,4-1          // point to last byte of rest
      add      edi,4-1
      rep      movsb
      cld
@@EXIT:
      pop      edi
      pop      esi
End;
```

ESBSet routine

Unit
ESBRtns

Declaration
```
Procedure ESBSet(var Dest; const Size: Integer);
```

Description
Fills given structure with specified number of $FF values, effectively setting it.

Implementation

```
procedure ESBSet (var Dest; const Size: Integer);
begin
   FillChar (Dest, Size, $FF);
End;
```

Ext2CEStr routine

Unit
ESBRtns

Declaration
```
Function Ext2CEStr(const E: Extended; const Decimals: Byte): String;
```

Description
Converts an Extended Real into an exact String, No padding, with given number of Decimal Places, with Commas separating thousands

Implementation

```
function Ext2CEStr (const E: Extended; const Decimals: Byte): String;
begin
   try
      Result := FloatToStrF (E, ffNumber, 18, Decimals)
   except
      Result := '';
   end;
End;
```

Ext2EStr routine

Unit
ESBRtns

Declaration
```
Function Ext2EStr(const E: Extended; const Decimals: Byte): String;
```

Description
Converts an Extended Real into an exact String, No padding, with given number of Decimal Places

Implementation

```
function Ext2EStr (const E: Extended; const Decimals: Byte): String;
begin
   try
      Result := FloatToStrF (E, ffFixed, 18, Decimals)
   except
      Result := '';
   end;
End;
```

Ext2EStr2 routine

Unit
ESBRtns

Declaration
```
Function Ext2EStr2(const E: Extended; const Decimals: Byte): String;
```

Description
Converts an Extended Real into an exact String, No padding, with at most the specified number of Decimal Places

Implementation

```
function Ext2EStr2 (const E: Extended; const Decimals: Byte): String;
begin
   Result := Ext2EStr (E, Decimals);
   Result := StripTChStr (Result, '0');
   if Length (Result) > 0 then
      if Result [Length (Result)] = DecimalSeparator then
         Result := LeftStr (Result, Length (Result) - 1);
End;
```

Ext2Str routine

Unit
ESBRtns

Declaration
```
Function Ext2Str(const E: Extended; const Len, Decimals: Byte): String;
```

Description
Converts an Extended Real into a String of specified Length, using NumPadCh for Left Padding, and with Specified number of Decimals

Implementation

```
function Ext2Str (const E: Extended; const Len, Decimals: Byte): String;
begin
   try
      Result := FloatToStrF (E, ffFixed, 18, Decimals)
   except
      Result := '';
   end;
   Result := PadChLeftStr (LeftStr (Result, Len), NumPadCh, Len);
End;
```

FillStr routine

Unit
ESBRtns

Declaration
```
Function FillStr(const Ch : Char; const N : Integer): string;
```

Description
Returns a string composed of N occurrences of Ch.

Implementation

```
function FillStr (const Ch : Char; const N : Integer): string;
begin
   SetLength (Result, N);
   FillChar (Result [1], N, Ch);
End;
```

FlipAllBits routine

Unit
ESBRtns

Declaration
```
Procedure FlipAllBits(var Body: TBitList);
```

Description
Flips all Bits in the BitList, i.e 1 -> 0 and 0 -> 1

Implementation

```
procedure FlipAllBits (var Body: TBitList);

begin
   Body:= Body xor $FFFF
End;
```

FlipAllLBits routine

Unit
ESBRtns

Declaration
```
Procedure FlipAllLBits(var Body: TLongBitList);
```

Description
Flips all Bits in a LongBitList, i.e 1 -> 0 and 0 -> 1

Implementation

```
procedure FlipAllLBits (var Body: TLongBitList);
begin
   Body:= Body xor $FFFFFFFF
End;
```

FlipBit routine

Unit
ESBRtns

Declaration
```
Procedure FlipBit(var Body: TBitList; const I: Byte);
```

Description
Flips specified Bit in the BitList, i.e. 0 -> 1 and 1 -> 0

Implementation

```
procedure FlipBit (var Body: TBitList; const I: Byte);

begin
   Body:= Body xor ($0001 shl I)
End;
```

FlipLBit routine

Unit
ESBRtns

Declaration
```
Procedure FlipLBit(var Body: TLongBitList; const I: Byte);
```

Description
Flips specified Bit in a LongBitList, i.e. 0 -> 1 and 1 -> 0

Implementation

```
procedure FlipLBit (var Body: TLongBitList; const I: Byte);
begin
   Body:= Body xor (TLongBitList (1) shl I)
End;
```

HashStr routine

Unit
ESBRtns

Declaration
```
Function HashStr(const N : Integer): string;
```

Description
Returns a string composed of N occurrences of '#'.

Implementation

```
function HashStr (const N : Integer): string;
begin
   Result := FillStr ('#', N);
End;
```

LBitIsSet routine

Unit
ESBRtns

Declaration
```
Function LBitIsSet(const Body: TLongBitList; const I: Byte): Boolean;
```

Description
Returns True if Specified Bit in a LongBitList is 1

Implementation

```
function LBitIsSet (const Body: TLongBitList; const I: Byte): Boolean;
begin
   Result := (Body and (TLongBitList (1) shl I)) <> 0
End;
```

LBits2Str routine

Unit
ESBRtns

Declaration
```
Function LBits2Str(const Body: TLongBitList): String32;
```

Description
Converts a Long Bit list to a string of '1' and '0'.

Implementation

```
function LBits2Str (const Body: TLongBitList): String32;
var
   I: Integer;
begin
   SetLength (Result, 32);
   for I := 0 to 32 do
      if LBitIsSet (Body, I) then
         Result [I + 1] := '1'
      else
         Result [I + 1] := '0';
End;
```

LeftAlignStr routine

Unit
ESBRtns

Declaration
```
Function LeftAlignStr(const S : string; const N : Integer): string;
```

Description
Returns a string of Length N with blank spaces added to the **end** of the string if S is too short, or returning the N Left-most characters of S if S is too long.

Implementation

```
function LeftAlignStr (const S : string; const N : Integer): string;
begin
   Result := PadRightStr (Copy (S, 1, N), N);
End;
```

LeftStr routine

Unit
ESBRtns

Declaration
```
Function LeftStr(const S : string; const N : Integer): string;
```

Description
Returns the substring consisting of the first N characters of S. If N > Length (S) then the substring = S.

Implementation

```
function LeftStr (const S : string; const N : Integer): string;
begin
   Result := Copy (S, 1, N);
End;
```

LeftTillStr routine

Unit
ESBRtns

Declaration
```
Function LeftTillStr(const S : string; const Ch : Char): string;
```

Description
Returns the substring consisting of the characters from S up to but not including the specified one. If the specified character is not found then a null string is returned.

Implementation

```
function LeftTillStr (const S : string; const Ch : Char): string;
var
   M: Integer;
begin
   M := Pos (Ch, S);
   if M < 2 then
      Result := ''
   else
      Result := Copy (S, 1, M - 1);
End;
```

LInt2CEStr routine

Unit
ESBRtns

Declaration
```
Function LInt2CEStr(const L : LongInt): string;
```

Description
Convert a LongInt into a Comma'ed String without Padding

Implementation

```
function LInt2CEStr (const L : LongInt): string;
var
   LS, L2, I : Integer;
   Temp : string;
begin
   Result := LInt2EStr (L);
   LS := Length (Result);
   L2 := (LS - 1) div 3;
   Temp := '';
   for I := 1 to L2 do
      Temp :=  ThousandSeparator + Copy (Result, LS - 3 * I + 1, 3) + Temp;
   Result := Copy (Result, 1, (LS - 1) mod 3 + 1) + Temp;
End;
```

LInt2CStr routine

Unit
ESBRtns

Declaration
```
Function LInt2CStr(const L : LongInt; const Len : Byte): string;
```

Description
Convert a LongInt into a Comma'ed String of length Len, with NumPadCh Padding to the Left

Implementation

```
function LInt2CStr (const L : LongInt; const Len : Byte): string;
begin
   Result := LInt2CEStr (L);
   Result := PadChLeftStr (LeftStr (Result, Len), NumPadCh, Len);
End;
```

LInt2EStr routine

Unit
ESBRtns

Declaration
```
Function LInt2EStr(const L: LongInt): String;
```

Description
Convert a LongInt into an exact String, No Padding

Implementation

```
function LInt2EStr (const L: LongInt): String;
begin
   try
      Result := IntToStr (L);
   except
      Result := '';
   end;
End;
```

LInt2Str routine

Unit
ESBRtns

Declaration
```
Function LInt2Str(const L: LongInt; const Len: Byte): String;
```

Description
Converts a LongInt into a String of length N withNumPadCh Padding to the Left

Implementation

```
function LInt2Str (const L: LongInt; const Len: Byte): String;
begin
   try
      Result := IntToStr (L);
   except
      Result := '';
   end;
   Result := PadChLeftStr (LeftStr (Result, Len), NumPadCh, Len);
End;
```

LInt2ZBEStr routine

Unit
ESBRtns

Declaration
```
Function LInt2ZBEStr(const L: LongInt): String;
```

Description
Convert a LongInt into an exact String, No Padding, with null returned if Value is 0

Implementation

```
function LInt2ZBEStr (const L: LongInt): String;
begin
   if L = 0 then
      Result := ''
   else
      try
         Result := IntToStr (L);
      except
         Result := '';
      end;
End;
```

LInt2ZBStr routine

Unit
ESBRtns

Declaration
```
Function LInt2ZBStr(const L: LongInt; const Len: Byte): String;
```

Description
Converts a LongInt into a String of length N withNumPadCh Padding to the Left, with blanks returned if Value is 0

Implementation

```
function LInt2ZBStr (const L: LongInt; const Len: Byte): String;
begin
   Result := LInt2ZBEStr (L);
   Result := PadChLeftStr (LeftStr (Result, Len), NumPadCh, Len);
End;
```

LInt2ZStr routine

Unit
ESBRtns

Declaration
```
Function LInt2ZStr(const L: LongInt; const Len: Byte): String;
```

Description
Converts a LongInt into a String of length N withNumPadCh Padding to the Left

Implementation

```
function LInt2ZStr (const L: LongInt; const Len: Byte): String;
begin
   Result := LInt2EStr (L);
   Result := PadChLeftStr (LeftStr (Result, Len), '0', Len);
End;
```

PadChLeftStr routine

Unit
ESBRtns

Declaration

```
Function PadChLeftStr(const S : string; const Ch : Char;   const Len :
Integer): string;
```

Description

Returns a string with specified characters added to the beginning of the string until the string is of the given length.

If Length (S) >= Len then NO padding occurs, and S is returned.

Implementation

```
function PadChLeftStr (const S : string; const Ch : Char;
   const Len : Integer): string;
var
   N: Integer;
begin
   N := Length (S);
   if N < Len then
      Result := FillStr (Ch, Len - N) + S
   else
      Result := S;
End;
```

PadChRightStr routine

Unit
ESBRtns

Declaration
```
Function PadChRightStr(const S : string; const Ch : Char;   const Len :
Integer): string;
```

Description
Returns a string with specified characters added to the end of the string until the string is of the given length.

If Length (S) >= Len then NO padding occurs, and S is returned.

Implementation

```
function PadChRightStr (const S : string; const Ch : Char;
   const Len : Integer): string;
var
   N: Integer;
begin
   N := Length (S);
   if N < Len then
      Result := S + FillStr (Ch, Len - N)
   else
      Result := S;
End;
```

PadLeftStr routine

Unit
ESBRtns

Declaration
```
Function PadLeftStr(const S : string; const Len : Integer): string;
```

Description
Returns a string with blank spaces added to the beginning of the string until the string is of the given length.

If Length (S) >= Len then NO padding occurs, and S is returned.

Implementation

```
function PadLeftStr (const S : string; const Len : Integer): string;
var
   N: Integer;
begin
   N := Length (S);
   if N < Len then
      Result := BlankStr (Len - N) + S
   else
      Result := S;
End;
```

PadRightStr routine

Unit
ESBRtns

Declaration
```
Function PadRightStr(const S : string; const Len : Integer): string;
```

Description
Returns a string with blank spaces added to the end of the string until the string is of the given length.

If Length (S) >= Len then NO padding occurs, and S is returned.

Implementation

```
function PadRightStr (const S : string; const Len : Integer): string;
var
   N: Integer;
begin
   N := Length (S);
   if N < Len then
      Result := S + BlankStr (Len - N)
   else
      Result := S;
End;
```

ReplaceChStr routine

Unit
ESBRtns

Declaration
```
Function ReplaceChStr(const S : string; const OldCh, NewCh : Char): string;
```

Description
Returns the String with all occurrences of OldCh character replaced with NewCh character.

Implementation

```
function ReplaceChStr (const S : string;
   const OldCh, NewCh : Char): string;
var
   I: Integer;
begin
   Result := S;
   if OldCh = NewCh then
      Exit;
   for I := 1 to Length (S) do
      if S [I] = OldCh then
         Result [I] := NewCh;
End;
```

ReverseBits routine

Unit
ESBRtns

Declaration
```
Procedure ReverseBits(var Body: TBitList);
```

Description
Reverses the Bit List, i.e. Bit 15 <-> Bit 0, Bit 14 <-> Bit1, etc.

Implementation

```
procedure ReverseBits (var Body: TBitList); assembler;
asm
      push esi
      push ebx

      mov  esi, eax
      mov  bx, Word Ptr [esi]
      sub   ax, ax       // clear ax for out going bit list
      mov   cx, 16       // 16 iterations needed for a word
      sub   dx, dx       // clear dx for additions

   @1:
      shl   ax, 1       // move all of ax right
      shr   bx, 1       // move lsb into CF
      adc   ax, dx       // add in the carry bit
      loop @1

      mov Word Ptr [esi], ax

      pop    ebx
      pop    esi
End;
```

RightAfterChStr routine

Unit
ESBRtns

Declaration
```
Function RightAfterChStr(const S : String; const Ch : Char): String;
```

Description
Returns the sub-string to the right AFTER the first ocurrence of specifiec character. If Ch not found then a Null String is returned.

Implementation

```
function RightAfterChStr (const S : String; const Ch : Char): String;
var
   M: Integer;
begin
   M := Pos (Ch, S);
   if M = 0 then
      Result := ''
   else
      Result := Copy (S, M + 1, Length (S) - M);
End;
```

RightAfterStr routine

Unit
ESBRtns

Declaration
```
Function RightAfterStr(const S : String; const N : Integer): String;
```

Description
Returns the sub-string to the right AFTER the first N Characters. if N >= Length (S) then a Null string is returned.

Implementation

```
function RightAfterStr (const S : String; const N : Integer): String;
begin
   Result := Copy (S, N + 1, Length (S) - N );
End;
```

RightAlignStr routine

Unit
ESBRtns

Declaration
```
Function RightAlignStr(const S : string; const N : Integer): string;
```

Description
Returns a string of Length N with blank spaces added to the **beginning** of the string if S is too short, or returning the N Left-most characters of S if S is too long.

Implementation

```
function RightAlignStr (const S : string; const N : Integer): string;
begin
   Result := PadLeftStr (Copy (S, 1, N), N);
End;
```

RightStr routine

Unit
ESBRtns

Declaration
```
Function RightStr(const S : string; const N : Integer): string;
```

Description
Returns the substring consisting of the last N characters of S. If N > Length (S) then the substring = S.

Implementation

```
function RightStr (const S : string; const N : Integer): string;
var
   M: Integer;
begin
   M := Length (S) - N + 1;
   if M < 1 then
      M := 1;
   Result := Copy (S, M, N);
End;
```

SetAllBits routine

Unit
ESBRtns

Declaration
```
Procedure SetAllBits(var Body: TBitList);
```

Description
Sets all Bits in the BitList to 1

Implementation

```
procedure SetAllBits (var Body: TBitList);

begin
   Body:= $FFFF
End;
```

SetAllLBits routine

Unit
ESBRtns

Declaration
```
Procedure SetAllLBits(var Body: TLongBitList);
```

Description
Sets all Bits in a LongBitList to 1

Implementation

```
procedure SetAllLBits (var Body: TLongBitList);
begin
   Body:= $FFFFFFFF
End;
```

SetBit routine

Unit
ESBRtns

Declaration
```
Procedure SetBit(var Body: TBitList; const I: Byte);
```

Description
Sets specified Bit in the BitList to 1

Implementation
```
procedure SetBit (var Body: TBitList; const I: Byte);

begin
   Body:= Body or ($0001 shl I)
End;
```

SetLBit routine

Unit
ESBRtns

Declaration
```
Procedure SetLBit(var Body: TLongBitList; const I: Byte);
```

Description
Sets specified Bit in a LongBitList to 1

Implementation

```
procedure SetLBit (var Body: TLongBitList; const I: Byte);
begin
   Body:= Body or (TLongBitList (1) shl I)
End;
```

Single2EStr routine

Unit
ESBRtns

Declaration
```
Function Single2EStr(const S: Single; const Decimals: Byte): String;
```

Description
Converts a Single Real into an exact String, No padding, with given number of Decimal Places

Implementation

```
function Single2EStr (const S: Single; const Decimals: Byte): String;
begin
   try
      Result := FloatToStrF (S, ffFixed, 7, Decimals)
   except
      Result := '';
   end;
End;
```

Single2Str routine

Unit
ESBRtns

Declaration
```
Function Single2Str(const S: Single; const Len, Decimals: Byte): String;
```

Description
Converts an Single Real into a String of specified Length, using NumPadCh for Left Padding, and with Specified number of Decimals

Implementation

```
function Single2Str (const S: Single; const Len, Decimals: Byte): String;
begin
   try
      Result := FloatToStrF (S, ffFixed, 7, Decimals)
   except
      Result := '';
   end;
   Result := PadChLeftStr (LeftStr (Result, Len), NumPadCh, Len);
End;
```

StarStr routine

Unit
ESBRtns

Declaration
```
Function StarStr(const N : Integer): string;
```

Description
Returns a string composed of N occurrences of '*'.

Implementation

```
function StarStr (const N : Integer): string;
begin
   Result := FillStr ('*', N);
End;
```

Str2Bits routine

Unit
ESBRtns

Declaration
```
Function Str2Bits(const S: String16): TBitList;
```

Description
Converts a string of '1' and '0' into a BitList.

Implementation

```
function Str2Bits (const S: String16): TBitList; assembler;
asm
      push esi
      push ebx
      mov   esi, eax

      lodsb          // Read Length
      sub   ah, ah
      mov   cx, ax       // & store in CX
      sub   bx, bx       // clear BX for bit list construction
      mov   dl, '0'       // for comparisons

   @1:   lodsb
      shl   bx, 1       // mov bx along
      cmp   al, dl
      je   @2
      add   bx, 1       // otherwise add 1
   @2:   loop @1;
      mov   ax, bx       // result must be in ax

      pop    ebx
      pop    esi
End;
```

Str2Byte routine

Unit
ESBRtns

Declaration
```
Function Str2Byte(const S: String): Byte;
```

Description
Converts a String into a Byte

Implementation

```
function Str2Byte (const S: String): Byte;
var
   L: LongInt;
begin
   L := Str2LInt (S);
   if L > High (Byte) then
      Result := High (Byte)
   else if L < Low (Byte) then
      Result := Low (Byte)
   else
      Result := L;
End;
```

Str2Ext routine

Unit
ESBRtns

Declaration
```
Function Str2Ext(const S: String): Extended;
```

Description
Converts a String into an Extended Real

Implementation

```
function Str2Ext (const S: String): Extended;
begin
   try
      Result := StrToFloat (S);
   except
      Result := 0;
   end;
End;
```

Str2Int routine

Unit
ESBRtns

Declaration
```
Function Str2Int(const S: String): Integer;
```

Description
Converts a String into an Integer

Implementation
```
function Str2Int (const S: String): Integer;
begin
   Result := Str2LInt (S);
End;
```

Str2LInt routine

Unit
ESBRtns

Declaration
```
Function Str2LInt(const S: String): LongInt;
```

Description
Converts a String into a LongInt

Implementation

```
function Str2LInt (const S: String): LongInt;
{$IFDEF Ver120}
var
   L: Int64;
{$ENDIF}
begin
   {$IFNDEF Ver120}
   try
      Result := StrToInt (S);
   except
      Result := 0;
   end;
   {$ELSE}
   try
      L := StrToInt64 (S);
      if L > High (LongInt) then
         Result := High (LongInt)
      else if L < Low (LongInt) then
         Result := Low (LongInt)
      else
         Result := L;
   except
      Result := 0;
   end;
   {$ENDIF}
End;
```

Str2SInt routine

Unit
ESBRtns

Declaration
```
Function Str2SInt(const S: String): ShortInt;
```

Description
Converts a String into a ShortInt

Implementation

```
function Str2SInt (const S: String): ShortInt;
var
   L: LongInt;
begin
   L := Str2LInt (S);
   if L > High (ShortInt) then
      Result := High (ShortInt)
   else if L < Low (ShortInt) then
      Result := Low (ShortInt)
   else
      Result := L;
End;
```

Str2SmallInt routine

Unit
ESBRtns

Declaration
```
Function Str2SmallInt(const S: String): SmallInt;
```

Description
Converts a String into a SmallInt

Implementation

```
function Str2SmallInt (const S: String): SmallInt;
var
   L: LongInt;
begin
   L := Str2LInt (S);
   if L > High (SmallInt) then
      Result := High (SmallInt)
   else if L < Low (SmallInt) then
      Result := Low (SmallInt)
   else
      Result := L;
End;
```

Str2Word routine

Unit
ESBRtns

Declaration
```
Function Str2Word(const S: String): Word;
```

Description
Converts a String into a Word

Implementation

```
function Str2Word (const S: String): Word;
var
   L: LongInt;
begin
   L := Str2LInt (S);
   if L > High (Word) then
      Result := High (Word)
   else if L < Low (Word) then
      Result := Low (Word)
   else
      Result := L;
End;
```

StripChStr routine

Unit
ESBRtns

Declaration
```
Function StripChStr(const S : string; const Ch : Char): string;
```

Description
Returns the String with all specified leading and trailing characters removed.

Implementation

```
function StripChStr (const S : string; const Ch: Char): string;
begin
   Result := StripTChStr (StripLChStr (S, Ch), Ch);
End;
```

StripLChStr routine

Unit
ESBRtns

Declaration
```
Function StripLChStr(const S : string; const Ch : Char): string;
```

Description
Returns the String with all specified leading characters removed.

Implementation

```
function StripLChStr (const S : string; const Ch: Char): string;
var
   I, Len: Integer;
begin
   Len := Length (S);
   I := 1;
   while (I <= Len) and (S [I] = Ch) do
      Inc (I);
   if (I > Len) then
      Result := ''
   else
      Result := Copy (S, I, Len - I + 1);
End;
```

StripTChStr routine

Unit
ESBRtns

Declaration
```
Function StripTChStr(const S : string; const Ch : Char): string;
```

Description
Returns the String with all specified trailing characters removed.

Implementation

```
function StripTChStr (const S : string; const Ch: Char): string;
var
   Len: Integer;
begin
   Len := Length (S);
   while (Len > 0) and (S [Len] = Ch) do
      Dec (Len);
   if Len = 0 then
      Result := ''
   else
      Result := Copy (S, 1, Len);
End;
```

TF2Boolean routine

Unit
ESBRtns

Declaration
```
Function TF2Boolean(const Ch : Char): Boolean;
```

Description
Converts a Character Value into its corresponding Boolean value:

'T', 't' -> True

Otherwise -> False

Implementation

```
function TF2Boolean (const Ch : Char): Boolean;
begin
   Result := Ch in ['T', 't'];
End;
```

YN2Boolean routine

Unit
ESBRtns

Declaration
```
Function YN2Boolean(const Ch : Char): Boolean;
```

Description
Converts a Character Value into its corresponding Boolean value:

'Y', 'y' -> True

Otherwise -> False

Implementation

```
function YN2Boolean (const Ch : Char): Boolean; assembler;
begin
   Result := Ch in ['Y', 'y'];
End;
```

String16 type

Unit
ESBRtns

Declaration
```
String16 = string [16];
```

String32 type

Unit
ESBRtns

Declaration
```
String32 = string [32];
```

TBitList type

Unit
ESBRtns

Declaration
```
TBitList = Word;
```

Description
Used for a Bit List of 16 bits from 15 -> 0

TLongBitList type

Unit
ESBRtns

Declaration
```
TLongBitList = LongInt;
```

Description
Used for a Bit List of 32 bits from 31 -> 0

NumPadCh global constant

Unit
ESBRtns

Declaration
```
NumPadCh : Char = ' ';
```

Description
Character to use for Left Hand Padding of Numerics