make.tol de Dct.Writer

Documentador de codigos que genera agendas de posts de otros programas desarrollados en lenguaje Tol y algunos otros lenguajes como Javascript, Html, Sql y Xml. El programa explota la riqueza de otros programas en comentarios, estructura y codigo para generar unos contenidos que son el paso previo para ser llevados a paginas web y generar con ellos documentacion en Pdf. Este codigo se le puede clasificar como metaprogramación. La metraprogramación consiste en escribir programas que escriben o manipulan otros programas o a si mismo. Usualmente no se ejecuta el fichero make.tol del documentador directamente, sino que es desde otros programas donde se invoca a su funcion principal DctMake(). Para invocar a DctMake() se le debe haber declarado previamente a) Text DctPre que es la ruta del directorio contenedor del directorio del programa a documentar y b) Set DctTre que es el arbol de ficheros a documentar.

Para utilizar toda la potencia de este codigo de autodocumentación Dct.Writer se han de tener en cuenta un conjunto de directrices, que afectan fundamentalmente a Tol, pero tambien a otros lenguajes de programacion sobre: a) el estilo de programacion, b) la forma de introducir los comentarios, c) las descripciones de las funciones y de las variables en Tol, etc. El nombre del programa es igual al de su directorio principal y bajo este directorio principal hay un subdirectorio dct para documentacion: a) donde se guardan las agendas de posts generadas, b) con los fichero Html de presentacion realizados a mano y c) las imagenes ilustrativas. Todas las agendas generadas, una por fichero del programa: a) se guardan en el directorio dct del programa y b) con el nombre del fichero con extension cambiando los puntos por subrayadores mas una nueva extension .age, por ejemplo, de txt.tol es dct/txt_tol.age, todo en minusculas.

Las presentaciones generadas a mano: a) se buscan en el directorio dct, b) el nombre del fichero de presentacion es el del nombre que presenta, incluida su extension, todo separado por subrayadores, con la extension .htm, por ejemplo, la presentacion de make.tol es dct/make_tol.htm, se conserva la antigua extension porque para agendas o presentaciones, si no se incluyentan, pueden coincidir, por ejemplo, las de los ficheros make.tol y make.bat que se diferencian al incluir su extension, asi es dct/make_tol.htm y dct/make_bat.htm, c) no es obligatorio incluir presentaciones ,htm manuales, solo se insertan si existe dicho fichero de presentacion y d) del fichero de presentacion se lee y se incluye todo el codigo html que se encuentra entre las etiquetas de inicio <body> y de fin </body>, esto permite, si se emplea el Css adecuado, reproducir el aspecto final de la pagina con sus imagenes y e) dentro de este codigo de presentacion no se emplean headers h1, h2 y h3 que se reservan para las secciones oficiales, si se han de simular titulos se ponen en un parrafo p con letra bold b o h4.

Las imagenes de las presentaciones manuales en html: a) estan en el directorio dct. b) se enlazan (src) como ../dct_writer/imagen.ext, siendo ext, por ejemplo, png. La elaboracion de la documentacion se realiza a partir de: a) la ruta previa al directorio del programa, por ejemplo, ../Web, ruta que puede ser relativa o absoluta, preferiblemente lo primero, y b) un arbol de entrada de descripcion de los ficheros a documentar. El arbol de entrada: a) contiene el nombre del programa y, por tanto, el nombre del directorio principal, asi, por ejemplo, de Omr.Forms y ../Web se forma ../Web/Omr.Forms/dct, b) todos los ficheros a documentar que pueden tener marcas que identifican al modulo principal y a su raiz [*] y a los fichero de salida

[>], el resto de ficheros se asumen de entrada, c) una breve descripcion que se utiliza para la generacion la estructura de ficheros y como descripcion si no hay otra y d) se documentan todos los ficheros, no se documentan los directorios, salvo el directorio principal cuya documentacion es la del modulo principal del programa, usualmente un make.tol

Este documentador distingue 4 tipos de ficheros: a) los ficheros en lenguaje Tol en codigo realzado, b) los ficheros en otros lenguajes de programacion en codigo realzado, c) los ficheros en texto plano, que no van en codigo realzado y d) los ficheros pdf que se dejan como estan en nombre y contenido. La documentación de un fichero Tol tienen las siguientes secciones: a) la Cabecera, cuyo contenido sale de la cabecera inicial del fichero Tol, b) la Presentacion, opcional, cuyo contenido proviene de un fichero Html generado de forma manual, que puede incluir imagenes, c) el Indice de declaraciones, donde cada declaracion enlaza con las variables y funciones que se definen a continuacion, esta seccion se subdivide en Inclusiones, Variables de control, Constantes, Funciones, etc. igual que los ficheros Tol, d) la secuencia de secciones de Inclusiones, Estructuras de datos, Variables de control, Constantes, Funciones de nombre corto, Funciones, Proceso, Pruebas y Finalizacion, esta secuencia es la misma que la del indice anterior y para cada fichero solo se ponen las que existen, e) el Codigo fuente completo con el contenido en codigo realzado de todo el fichero y f) el Arbol de estructura de ficheros, que es una representacion en Html del arbol que se recibe como entrada al documentador, en cada arbol se destaca el propio fichero frente al resto, este arbol enlaza con links a todos los ficheros del arbol del programa, esta seccion se pone al final porque al ser similar para todos los ficheros si se pone al principio a modo de menu de navegacion todas las paginas parecen inicialmente similares.

Los ficheros en lenguajes diferentes a Tol, por ejemplo, los de SQL, Javascript, Html, Xml, de mandatos, etc. tienen las siguientes secciones: a) la Cabecera, cuyo contenido puede salir de sus comentarios de cabecera y en otro caso sale de la descripcion del arbol, b) la Presentacion, que es opcional, c) el Codigo fuente completo con el contenido en codigo realzado de todo el fichero y b) el Arbol de estructura de ficheros. Este documentador asume que existen 2 tipos de links a las secciones: a) el link local, #, formado por el nombre del programa, el del fichero y el nombre de la seccion, este link local podria ser mas simple, con la seccion bastaria, pero se necesita todo porque cada seccion puede salir a formar parte de otras paginas compartiendo espacio con otras secciones con el mismo nombre de seccion o de fichero y b) el link global formado por ../programa/fichero.html#link.local. Sin embargo, habiendo realizado pruebas en FireFox, Chrome e IExplorer los links globales funcionan como locales, sin recargar la pagina, cuando se les invoca de forma local por lo que solo se plantea un tipo que es el link global con la forma ../programa/fichero.html#seccion. Por lo que ambos metodos, links globales y locales o solo globales, funcionan bien.

Este documentador genera 2 tipos de etiquetas para las secciones que coinciden con el titulo de la seccion: a) la etiqueta o titulo local que es solo el nombre de la seccion, en Tol, en variables y funciones esto coincide con el nombre de la variable o de la funcion mas () y b) la etiqueta o titulo global formado por etiqueta.local de programa, en este caso utilizar la etiqueta global como local resulta reiterado ya que, por ejemplo, 12 funciones seguidas irian todas con el funX() de Omr.Forms, funY() de Omr.Forms, etc., Es por ello que en este documentador se unifican las etiquetas de la siguiente forma: a) titulo ventana, enlaces y headers globales = seccion de programa, donde seccion es un nombre castellano o un nombre de variable o funcion, b) etiqueta a names, enlaces y headers locales que es solo la seccion. Esta diferencia entre etiquetas se podria crear en una fase posterior pero este

sección Esta diferencia entre etiquetas se podría crear en una fase posterior, pero este documentador la genera porque dispone de toda la informacion para crearlas evitandole esta tarea a procesos posteriores. La informacion que este documentador obtiene de la cabecera de un fichero de codigo es: a) fichero, file, b) autor, author, c) clases, classes y d) proposito, purpose.

Árbol de ficheros

Dct.Writer documenta programas Tol generado agendas para Html

- ← make.tol proceso principal de documentación de programas Tol
- tol directorios de código Tol
  - cmm funciones comunes
    - ← txt.tol de textos
    - ← any.tol de anything, cualquier gramática
    - ← fil.tol de de ficheros
    - ← htm.tol para generar lenguaje Html
  - app funciones especificas de aplicacion
    - ← shi.tol sintaxis realzada de código Tol, Xml, Html, etc.
    - ← dtr.tol de árboles de ficheros para documentación
    - ← dct.tol de documentación de código en Html
  - ← inc.tol para la inclusion de ficheros Tol
- dct donde se generan las agenda de post de documentación
  - → ejemplos.age ejemplo de una de las agendas generadas
- → arbol.html ejemplo de árbol para documentar codigo fuente
- → dct_writer.pdf documento resumen de funciones del programa

Declaraciones

Inclusiones

- `Set allInc`

  Inclusion de las funciones comunes.

Proceso

- `Real makDct`

  Con DctPre, ruta del directorio contenedor del directorio del programa, y DctTre, arbol de ficheros a documentar, crea la documentacion de un programa Tol.

Set allInc

```
////////////////////////////////////////////////////////////////////////////
Set  allInc = Include("tol/inc.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Inclusion de las funciones comunes.", allInc);
```

```
////////////////////////////////////////////////////////////////////////////
```

Real makDct

```
////////////////////////////////////////////////////////////////////////////
Real makDct = Case(
  ! ObjectExist("Text","DctPre"),
    { Text WriteLn("Error: sin definir la ruta Text DctPre"); FALSE },
  ! ObjectExist("Set","DctTre"),
    { Text WriteLn("Error: sin definir el arbol Set DctTre"); FALSE },
  TRUE, DctMake(DctPre, DctTre));
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Con DctPre, ruta del directorio contenedor del directorio del programa, y
DctTre, arbol de ficheros a documentar, crea la documentacion de un programa
Tol.",
makDct);
////////////////////////////////////////////////////////////////////////////
```

txt.tol de Dct.Writer

Funciones de textos.

Declaraciones

Funciones de nombre corto

- `Text Q(Text txtVal)`
  Retorna un texto entre dobles comillas. Equivalente a la funcion Tol Qt().
- `Text F(Anything anyVal)`
  Retorna numeros, fechas, textos, conjuntos como un texto de formato simple.

Funciones

- `Real TxtBeginStrict(Text txtInp, Text txtIni)`
  Version estricta de TextBeginWith(), da falso siempre que txtIni sea nulo.
- `Text TxtBetween2Tag(Text inpTxt, Text tagIni, Text tagEnd, Real cmpFlg)`
  Retorna un subtexto entre la primera ocurrencia de tagIni y tagEnd. Si tagIni o tagEnd no aparecen retorna la tira vacia. Si cmpFlg es cierto entonces aplica la funcion Compact() al texto que retorna. Por ejemplo: TxtBetween2Tag('a b [[c]] d [[e]] f', '[[', ']]', TRUE) retorna 'c'.
- `Text TxtCat(Text iniTxt, Text sepTxt, Text endTxt)`
  Retorna un texto resultado de concatenar 2 textos con un separador en medio, si alguno de los 2 textos son la tira vacia retorna el otro texto.
- `Real TxtEndStrict(Text txtInp, Text txtEnd)`
  Version estricta de TextEndAt(), da falso siempre que txtEnd sea nulo.
- `Set TxtForChr(Text inpTxt, Code funChr)`
  Retorna el conjunto resultado de aplicar la funcion funChr(Text oneChr) a todos los caracteres del texto de entrada txtInp. Retorna un Set a imagen de las funciones basicas For() y EvalSet().
- `Set TxtLineWrap(Text txtInp, Real linMax, Real cmpCtr)`
  Retorna un conjunto de 2 texto el primero con un máximo de linMax caracteres y el segundo con el resto. Es el resultado de cortar txtInp por el primer blanco que permita que el corte cumpla la condición inicial. Si el texto de entrada es mas corte que linMax retorna un conjunto formado por el texto inicial y la tira vacia. Si el corte es imposible busca el mejor corte posible y si no lo encuentra retorna un conjunto formado por el texto inicial y la tira vacia. Si cmpCtr es true los resultados son compactados. Tambien existe en Tol la funcion Wrap() con ciertas semejanzas, aunque mas a TxtParagraphWrap().
- `Set TxtSplitBy1Tag(Text txtInp, Text tagBrk)`

Retorna un conjunto de textos resultado de cortar el texto de entrada por un unico tag tagBrk incluyendo dicho tagBrk al inicio y al final de cada texto que enmarca, las ocurrencias impares de tagBrk al inicio del texto y las impares al final. Tiene mas sentido cuando el numero de ocurrencias de tagBrk es par. Si el numero de ocurrencias de tagbrk es impar funciona como si al final del texto txtInp hubiera una ultima ocurrencia. Fue una funcion recursiva, pero la versión 2.0.1 de Tol se caia con textos grandes, ahora es una funcion iterativa. TxtSplitBy1Tag(aaa|::|bbb|---|ccc, |) -> [aaav, |::|, bbb, |---|, ccc].

- `Set TxtSplitBy2Fast(Text txtInp, Text tagIni, Text tagEnd)`
  Retorna un conjunto de textos resultado de cortar el texto de entrada por dos tags de inicio y de fin incluyendo dichos tags al inicio y al final de cada texto que enmarcan. Los tags tagIni y tabEnd no han de ser nulos ni iguales, si son iguales lo correcto seria utilizar la funcion TxtSplitBy1Tag(). No es una funcion recursiva, es mas rapida que TxtSplitBy2Tag() pero menos resistente a la reiteracion de tags de inicio con un unico final. No funciona correctamente si los tags inicial y final son iguales. Tiene mas sentido cuando a cada tag de inicio le corresponde uno de final. TxtSplitBy2Fast(aaa<::>bbb<--->ccc, <, >) -> [aaav, <::>, bbb, <--->, ccc].
- `Set TxtSplitBy2Tag(Text txtInp, Text tagIni, Text tagEnd)`
  Retorna un conjunto de textos resultado de cortar el texto de entrada por un dos tags de inicio y de fin incluyendo dichos tags al inicio y al final de cada texto que enmarcan. Los tags tagIni y tabEnd no han de ser nulos. Tiene mas sentido cuando a cada tag de inicio le corresponde uno de final. Es una funcion recursiva. Si funciona correctamente si los tags inicial y final son iguales. TxtSplitBy2Tag(aaa<::>bbb<--->ccc, <, >) -> [aaav, <::>, bbb, <--->, ccc].
- `Set TxtTokenizer(Text txtInp, Text tagBrk)`
  Retorna un conjunto de textos resultado de cortar el texto de entrada por un unico tag tagBrk no incluyendo el tag tagBrk dentro de los textos. Se soporta en la funcion Tol Tokenizer() que rompe por un unico caracter. Usa como caracter interno de corte el 7 (bell), esperando que no aparezca.

Funciones de nombre corto

Text Q()

```
////////////////////////////////////////////////////////////////////////////
Text Q(Text txtVal) // Text
////////////////////////////////////////////////////////////////////////////
{ Char(34) + txtVal + Char(34) };
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un texto entre dobles comillas. Equivalente a la funcion Tol Qt().",
Q);
////////////////////////////////////////////////////////////////////////////
```

Text F()

```
////////////////////////////////////////////////////////////////////////////
Text F(Anything anyVal)
////////////////////////////////////////////////////////////////////////////
```

```
{
  Text graVal = Grammar(anyVal);
  Case
  (
    graVal=="Text", anyVal, // Ya es texto

    graVal=="Real", If(EQ(anyVal, Round(anyVal)),
                       FormatReal(anyVal, "%.0lf"),  // Entero sin decimales
                       FormatReal(anyVal, "%.2lf")), // 2 decimales

    graVal=="Date", If(Hour(anyVal),
                       FormatDate(anyVal, "%c%Y/%m/%d %h:%i:%s"), // Tiempo
                       FormatDate(anyVal, "%c%Y/%m/%d")),         // Fecha

    graVal=="Set",
    {
      Real crdSet = Card(anyVal);
      Case(
      EQ(crdSet,0), "[]",
      EQ(crdSet,1), "["+F(anyVal[1])+"]",
      TRUE,         "["+F(anyVal[1])+SetSum(For(2,crdSet,Text(Real setPos)
                  { "|"+F(anyVal[setPos]) }))+"]")
    },

    TRUE,           "Not basic type"
  )
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna numeros, fechas, textos, conjuntos como un texto de formato simple.",
F);
////////////////////////////////////////////////////////////////////////////
```

Real TxtBeginStrict()

```
////////////////////////////////////////////////////////////////////////////
Real TxtBeginStrict(Text txtInp, // Texto de entrada
                    Text txtIni) // Texto inicial
////////////////////////////////////////////////////////////////////////////
{ If(txtIni=="", FALSE, TextBeginWith(txtInp, txtIni)) };
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Version estricta de TextBeginWith(), da falso siempre que txtIni sea nulo.",
TxtBeginStrict);
////////////////////////////////////////////////////////////////////////////
```

Text TxtBetween2Tag()

```
////////////////////////////////////////////////////////////////////////////
Text TxtBetween2Tag(Text inpTxt, // Texto de entrada
                    Text tagIni, // Tag inicial
                    Text tagEnd, // Tag final
                    Real cmpFlg) // Si true aplica Compact()
////////////////////////////////////////////////////////////////////////////
{
  Real posIni = TextFind(inpTxt, tagIni);
  Text result = If(LE(posIni,0), "",
  {
    Real lenIni = TextLength(tagIni);
    Real posSub = posIni + lenIni;
    Real posEnd = TextFind(inpTxt, tagEnd, posSub);
    If(LE(posEnd, 0), "", Sub(inpTxt, posSub, posEnd-1))
  });
  If(cmpFlg, Compact(result), result)
};
////////////////////////////////////////////////////////////////////////////
```

```
PutDescription(
"Retorna un subtexto entre la primera ocurrencia de tagIni y tagEnd.
Si tagIni o tagEnd no aparecen retorna la tira vacia.
Si cmpFlg es cierto entonces aplica la funcion Compact() al texto que retorna.
Por ejemplo:
  TxtBetween2Tag('a b [[ c ]] d [[ e ]] f', '[[', ']]', TRUE)
  retorna 'c'.",
TxtBetween2Tag);
////////////////////////////////////////////////////////////////////////////
```

Text TxtCat()

```
////////////////////////////////////////////////////////////////////////////
Text TxtCat(Text iniTxt, // Texto inicial
            Text sepTxt, // Texto de separacion
            Text endTxt) // Texto final
////////////////////////////////////////////////////////////////////////////
{
  Case(
    iniTxt=="", endTxt,
    endTxt=="", iniTxt,
    TRUE,       iniTxt + sepTxt + endTxt)
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un texto resultado de concatenar 2 textos con un separador en medio,
si alguno de los 2 textos son la tira vacia retorna el otro texto.",
TxtCat);
////////////////////////////////////////////////////////////////////////////
```

Real TxtEndStrict()

```
////////////////////////////////////////////////////////////////////////////
Real TxtEndStrict(Text txtInp, // Texto de entrada
                  Text txtEnd) // Texto final
////////////////////////////////////////////////////////////////////////////
{ If(txtEnd=="", FALSE, TextEndAt(txtInp, txtEnd)) };
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Version estricta de TextEndAt(), da falso siempre que txtEnd sea nulo.",
TxtEndStrict);
////////////////////////////////////////////////////////////////////////////
```

Set TxtForChr()

```
////////////////////////////////////////////////////////////////////////////
Set TxtForChr(Text inpTxt, // Texto de entrada
              Code funChr) // Funcion tipo Anything(Text oneChr)
////////////////////////////////////////////////////////////////////////////
{
  Real lenTxt = TextLength(inpTxt);
  For(1, lenTxt, Anything(Real posTxt) { funChr(Sub(inpTxt,posTxt,posTxt)) })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el conjunto resultado de aplicar la funcion funChr(Text oneChr)
a todos los caracteres del texto de entrada txtInp.
Retorna un Set a imagen de las funciones basicas For() y EvalSet().",
TxtForChr);
////////////////////////////////////////////////////////////////////////////
```

Set TxtLineWrap()

```
////////////////////////////////////////////////////////////////////////////
Set  TxtLineWrap(Text txtInp, // Texto de entrada
                 Real linMax, // Maximo numero de caracteres por linea
                 Real cmpCtr) // Si true entonces compacta
////////////////////////////////////////////////////////////////////////////
{
  Text txtCmp = If(cmpCtr, Compact(txtInp), txtInp);
  Text txtRev = Reverse(txtCmp);
  Real txtLen = TextLength(txtCmp);
  Set  cutSet = If(LE(txtLen, linMax), [[txtCmp, ""]], // Ya esta hecho
  {
    Real blkPos = TextFind(txtRev, " ", txtLen-linMax); // Busca para atras

    If(GE(blkPos, 1),
    {
      SetOfText(Sub(txtCmp, 0,               txtLen-blkPos),
                Sub(txtCmp, txtLen-blkPos+1, txtLen))
    },
    {
      // No se puede cortar
      Real blkBad = TextFind(txtCmp, " ", linMax+1); // Busca hacia adelante

      If(LT(blkBad, 0), [[txtCmp, ""]], // No hay corte posible
      {
        SetOfText(Sub(txtCmp, 0,        blkBad-1), // Hay un mal corte
                  Sub(txtCmp, blkBad+1, txtLen))
      })
    })
  });
  If(cmpCtr, SetOfText(Compact(cutSet[1]),Compact(cutSet[2])), cutSet)
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto de 2 texto el primero con un máximo de linMax caracteres
y el segundo con el resto.
Es el resultado de cortar txtInp por el primer blanco que permita que el corte
cumpla la condición inicial.
Si el texto de entrada es mas corte que linMax retorna un conjunto formado
por el texto inicial y la tira vacia.
Si el corte es imposible busca el mejor corte posible y si no lo encuentra
retorna un conjunto formado por el texto inicial y la tira vacia.
Si cmpCtr es true los resultados son compactados.
Tambien existe en Tol la funcion Wrap() con ciertas semejanzas,
aunque mas a TxtParagraphWrap().",
TxtLineWrap);
////////////////////////////////////////////////////////////////////////////
```

Set TxtSplitBy1Tag()

```
////////////////////////////////////////////////////////////////////////////
Set TxtSplitBy1Tag(Text txtInp, // Texto de entrada
                   Text tagBrk) // Tag por el que se corta
////////////////////////////////////////////////////////////////////////////
{
  Set txtTok = TxtTokenizer(txtInp, tagBrk);
  For(1, Card(txtTok), Text(Real posTok) // Ciclo para impares y pares
  { If(posTok%2, txtTok[posTok], tagBrk+txtTok[posTok]+tagBrk) })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto de textos resultado de cortar el texto de entrada por un
unico tag tagBrk incluyendo dicho tagBrk al inicio y al final de cada texto
que enmarca, las ocurrencias impares de tagBrk al inicio del texto y
las impares al final.
Tiene mas sentido cuando el numero de ocurrencias de tagBrk es par.
Si el numero de ocurrencias de tagbrk es impar funciona como si al final del
texto txtInp hubiera una ultima ocurrencia.
Fue una funcion recursiva, pero la versión 2.0.1 de Tol se caia con textos
grandes, ahora es una funcion iterativa.
TxtSplitBy1Tag(aaa|::|bbb|---|ccc, |) -> [aaav, |::|, bbb, |---|, ccc].",
TxtSplitBy1Tag);
```

```
////////////////////////////////////////////////////////////////////////////
```

Set TxtSplitBy2Fast()

```
////////////////////////////////////////////////////////////////////////////
Set TxtSplitBy2Fast(Text txtInp, // Texto de entrada
                    Text tagIni, // Tag inicial por el que se corta
                    Text tagEnd) // Tag final por el que se corta
////////////////////////////////////////////////////////////////////////////
{
  Text chrBrk = Char(7); // Caracter auxiliar de corte que se espera unico
  Text repEnd = Replace(txtInp, tagEnd, tagEnd+chrBrk);
  Set  txtSet = Tokenizer(repEnd, chrBrk);

  Set  cicSet = EvalSet(txtSet, Set(Text txtTok)
  {

    Text repIni = Replace(txtTok, tagIni, chrBrk+tagIni);
    Set  tokSet = Tokenizer(repIni, chrBrk);
    Select(tokSet, Real(Text tokTxt) { tokTxt != "" }) // Elimina los vacios
  });
  BinGroup("<<", cicSet) // De conjunto de pares a conjunto lineal
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto de textos resultado de cortar el texto de entrada por
dos tags de inicio y de fin incluyendo dichos tags al inicio y al final de
cada texto que enmarcan.
Los tags tagIni y tabEnd no han de ser nulos ni iguales, si son iguales lo
correcto seria utilizar la funcion TxtSplitBy1Tag().
No es una funcion recursiva, es mas rapida que TxtSplitBy2Tag() pero menos
resistente a la reiteracion de tags de inicio con un unico final.
No funciona correctamente si los tags inicial y final son iguales.
Tiene mas sentido cuando a cada tag de inicio le corresponde uno de final.
TxtSplitBy2Fast(aaa<::>bbb<--->ccc, <, >) -> [aaav, <::>, bbb, <--->, ccc].",
TxtSplitBy2Fast);
////////////////////////////////////////////////////////////////////////////
```

Set TxtSplitBy2Tag()

```
////////////////////////////////////////////////////////////////////////////
Set TxtSplitBy2Tag(Text txtInp, // Texto de entrada
                   Text tagIni, // Tag inicial por el que se corta
                   Text tagEnd) // Tag final por el que se corta
////////////////////////////////////////////////////////////////////////////
{
  Real posIni = TextFind(txtInp, tagIni);
  If(LE(posIni,0), SetOfText(txtInp), // Nada que cortar
  {
    Real lenTxt = TextLength(txtInp); // Longitud del texto de entrada
    Real lenIni = TextLength(tagIni); // Longitud del tag inicial
    Real lenEnd = TextLength(tagEnd); // Longitud del tag final
    Real posSub = posIni + lenIni;
    Real posEnd = TextFind(txtInp, tagEnd, posSub);
    Case(
      Or(And(EQ(posIni,1),LE(posEnd,0)), // Ini en posicion 1 pero no termina
         And(EQ(posIni,1),EQ(posEnd+lenEnd-1,lenTxt))), // Justo ini y final
         SetOfText(txtInp),
      Or(And(GT(posIni,1),LE(posEnd,0)), // Ini en posicion >1 pero no termina
         And(GT(posIni,1),EQ(posEnd+lenEnd-1,lenTxt))), // Justo al final
         SetOfText(
           Sub(txtInp, 1,      posIni-1),
           Sub(txtInp, posIni, lenTxt)),
      And(EQ(posIni,1),GT(posEnd,1)), // Inicia en posicion 1, termina y sigue
        SetOfText(
          Sub(txtInp, 1,       posEnd+lenEnd-1)) <<
         TxtSplitBy2Tag(Sub(txtInp, posEnd+lenEnd, lenTxt), tagIni, tagEnd),
      TRUE,                             // Inicia posicion >1, termina y sigue
```

```
        SetOfText(
          Sub(txtInp, 1,       posIni-1),
          Sub(txtInp, posIni,  posEnd+lenEnd-1)) <<
          TxtSplitBy2Tag(Sub(txtInp, posEnd+lenEnd, lenTxt), tagIni, tagEnd))
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto de textos resultado de cortar el texto de entrada por un
dos tags de inicio y de fin incluyendo dichos tags al inicio y al final de
cada texto que enmarcan.
Los tags tagIni y tabEnd no han de ser nulos.
Tiene mas sentido cuando a cada tag de inicio le corresponde uno de final.
Es una funcion recursiva.
Si funciona correctamente si los tags inicial y final son iguales.
TxtSplitBy2Tag(aaa<::>bbb<--->ccc, <, >) -> [aaav, <::>, bbb, <--->, ccc].",
TxtSplitBy2Tag);
////////////////////////////////////////////////////////////////////////////
```

Set TxtTokenizer()

```
////////////////////////////////////////////////////////////////////////////
Set TxtTokenizer(Text txtInp, // Texto de entrada
                 Text tagBrk) // Tag por el que se corta
////////////////////////////////////////////////////////////////////////////
{ Tokenizer(Replace(txtInp, tagBrk, Char(7)), Char(7)) };
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto de textos resultado de cortar el texto de entrada por un
unico tag tagBrk no incluyendo el tag tagBrk dentro de los textos.
Se soporta en la funcion Tol Tokenizer() que rompe por un unico caracter.
Usa como caracter interno de corte el 7 (bell), esperando que no aparezca.",
TxtTokenizer);
////////////////////////////////////////////////////////////////////////////
```

any.tol de Dct.Writer

Funciones de anything.

Declaraciones

Funciones de nombre corto

- `Anything A(Anything anyVal, Text msgVal)`
  Retorna el valor que recibe como parametro visualizandolo antes. Para la visualizacion antepone al valor el mensaje mensaje msgVal. Se emplea para realizar una asignacion comunicada.

Funciones de nombre corto

Anything A()

```
////////////////////////////////////////////////////////////////////////////
Anything A(Anything anyVal, // Cualquier calor
           Text     msgVal) // Mensaje de asignacion
////////////////////////////////////////////////////////////////////////////
{
  Text WriteLn(msgVal+": "+F(anyVal));
  anyVal
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el valor que recibe como parametro visualizandolo antes.
Para la visualizacion antepone al valor el mensaje mensaje msgVal.
Se emplea para realizar una asignacion comunicada.",
A);
////////////////////////////////////////////////////////////////////////////
```

fil.tol de Dct.Writer

Funciones de basicas de ficheros.

Declaraciones

Funciones

- `Text FilBetween2Tag(Text filPth, Text tagIni, Text tagEnd)`
  Retorna el codigo resultado de reconvertir &, < y >.
- `Text FilDelExtLow(Text namFil)`
  Retorna un nombre de fichero cambiando el . de la extension y cualquier otro punto por un subrayador y pasandolo a minusculas.
- `Real FilTextFind(Text filPth, Text fndTxt)`
  Retorna la posicion de un texto dentro de un fichero Es similar a TextFind() pero para ficheros.

Text FilBetween2Tag()

```
////////////////////////////////////////////////////////////////////////////
Text FilBetween2Tag(Text filPth, // Ruta de un fichero
                    Text tagIni, // Tag inicial
                    Text tagEnd) // Tag final
////////////////////////////////////////////////////////////////////////////
{
  If(!FileExist(filPth), "", // Nada si el fichero no existe
  If(And(tagIni=="", tagEnd==""),
     ReadFile(filPth), // Todo el fichero
     TxtBetween2Tag(ReadFile(filPth), tagIni, tagEnd, FALSE))) // No compacta
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo resultado de reconvertir &, < y >.",
FilBetween2Tag);
////////////////////////////////////////////////////////////////////////////
```

Text FilDelExtLow()

```
////////////////////////////////////////////////////////////////////////////
Text FilDelExtLow(Text namFil)
////////////////////////////////////////////////////////////////////////////
{ Replace(ToLower(namFil),".","_") };
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un nombre de fichero cambiando el . de la extension y cualquier otro
punto por un subrayador y pasandolo a minusculas.",
FilDelExtLow);
////////////////////////////////////////////////////////////////////////////
```

Real FilTextFind()

```
////////////////////////////////////////////////////////////////////////////
Real FilTextFind(Text filPth, // Ruta de un fichero
                 Text fndTxt) // Texto a buscar
////////////////////////////////////////////////////////////////////////////
{ If(!FileExist(filPth), 0, TextFind(ReadFile(filPth), fndTxt)) };
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna la posicion de un texto dentro de un fichero
Es similar a TextFind() pero para ficheros.",
FilDelExtLow);
////////////////////////////////////////////////////////////////////////////
```

htm.tol de Dct.Writer

Funciones de basicas de Html (HyperText Markup Language).

Declaraciones

Funciones

- `Text HtmAsc2Xml(Text codInp)`
  Retorna el codigo resultado de reconvertir &, < y >.
- `Text Htm2Ide(Text prgNam, Text lblTxt, Real makLow)`
  Retorna una etiqueta de texto convertida en identificador Html, por ejemplo, para utilizarse en un <a name>. Un idetificador ha de empezar por letras A..Z o a..z y seguir por letras, numeros 0..9 o subrayador, menos, dos puntos o punto. Esta funcion solo conserva las letras y numeros en su posicion y cualquier otro caracter lo cambia por el subrayador. Dependiendo del parametro makLow lo pasa todo a minusculas. Si no empieza por letra le antepone una X.
- `Text HtmPutLinks(Text codTxt, Set lnkTab, Text opeSep)`
  Retorna un codigo Html resultado de ponerle links a determinados palabras que son identificadores. Los identificadores se delimitan por determinados caracteres, por ejemplo, blanco, coma, punto, 2 puntos en el caso del lenguaje escrito, mas, menos, slash en el caso de matematicas o codigo. De esta forma si recibe como identificador agua no se lo pone a paraguas. En sensible a mayusculas y minusculas. Los operadores se reciben como un texto y cada caracter es un operador. Si el texto de operadores es vacio asume una lista estandar. Si la tabla de etiquetas y links es vacia, no hace nada.
- `Text HtmPutCharts(Text codTxt)`
  Si hay graficos online inserta las instrucciones para que se vean. Si no hay graficos online retorna el mismo codigo Html que recibe como parametro.

Text HtmAsc2Xml()

```
////////////////////////////////////////////////////////////////////////////
Text HtmAsc2Xml(Text codInp) // Codigo de entrada
////////////////////////////////////////////////////////////////////////////
{
  ReplaceTable(codInp, [[ [["&", "&amp;"]],
                          [["<", "&lt;"]],
                          [[">", "&gt;"]] ]], 1)
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo resultado de reconvertir &, < y >.",
HtmAsc2Xml);
////////////////////////////////////////////////////////////////////////////
```

Text Htm2Ide()

```
////////////////////////////////////////////////////////////////////////////
Text Htm2Ide(Text prgNam, // Nombre del programa
             Text lblTxt, // Etiqueta de entrada
```

```
            Real makLow) // A minusculas
////////////////////////////////////////////////////////////////////////////
{
  Text lblInp = ReplaceTable(prgNam+"_"+lblTxt, [[
    [["á","a"]],[["é","e"]],[["í","i"]],[["ó","o"]],[["ú","u"]],[["ü","u"]],
    [["Á","A"]],[["É","E"]],[["Í","I"]],[["Ó","O"]],[["Ú","U"]],[["Ü","U"]],
    [["ñ","n"]],[["Ñ","N"]]
  ]], 1);

  Set  setChr = TxtForChr(lblInp, Text(Text oneChr)
  {
    Case(
      And(oneChr >= "0", oneChr <= "9"), oneChr,
      And(oneChr >= "A", oneChr <= "Z"), If(makLow, ToLower(oneChr), oneChr),
      And(oneChr >= "a", oneChr <= "z"), oneChr,
      TRUE,                              "_")
  });
  Text txtSum = If(Card(setChr), SetSum(setChr), "");

  If(txtSum=="",  "",           // No hay identificador
  If(txtSum<"A",  "X"+txtSum,   // No ha de empezar por numero
  If(txtSum=="_", "X"+txtSum,   // No ha de empezar por _
                  txtSum)))  // Es correcto
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna una etiqueta de texto convertida en identificador Html, por ejemplo,
para utilizarse en un <a name></a>.
Un idetificador ha de empezar por letras A..Z o a..z y seguir por letras,
numeros 0..9 o subrayador, menos, dos puntos o punto.
Esta funcion solo conserva las letras y numeros en su posicion y cualquier
otro caracter lo cambia por el subrayador.
Dependiendo del parametro makLow lo pasa todo a minusculas.
Si no empieza por letra le antepone una X.",
Htm2Ide);
////////////////////////////////////////////////////////////////////////////
```

Text HtmPutLinks()

```
////////////////////////////////////////////////////////////////////////////
Text HtmPutLinks(Text codTxt, // Codigo Html de entrada
                 Set  lnkTab, // Tabla de etiquetas y enlaces
                 Text opeSep) // Operadores de 1 char que separan etiquetas
////////////////////////////////////////////////////////////////////////////
{
  If(!Card(lnkTab), codTxt,
  {
    Text opeDef = " \n"+Char(9)+Char(34)+ // blanco, cr/lf, tab, comilla doble
                  "¡!$%&()*+,-/:;<=>¿?[]^{|}~";  // sin comilla simple
    Text opeTxt = If(opeSep!="", opeSep, opeDef); // Si no hay los por defecto

    Text opeGen = Char(64); // Bell, el operador generico

    Set  opeRep = For(1, TextLength(opeTxt), Set(Real opePos)
    { // Tabla de remplazamiento de operadores
      Text oneChr = Sub(opeTxt,opePos, opePos);
      SetOfText(oneChr, opeGen+oneChr+opeGen)
    });

    Text codPar = ReplaceTable(codTxt, opeRep, 1); // Parse bruto

    Set  codRep = EvalSet(lnkTab, Set(Set codRow)  // Tabla reemplazamiento
                    { SetOfText(opeGen+codRow[1]+opeGen, codRow[2]) });

    Text codMrk = ReplaceTable(codPar, codRep, 1); // Inyecta enlaces
    Text codCls = Replace(codMrk, opeGen, "");  // Limpiar operador generico

    codCls
  })
};
////////////////////////////////////////////////////////////////////////////
```

```
PutDescription(
"Retorna un codigo Html resultado de ponerle links a determinados palabras que
son identificadores.
Los identificadores se delimitan por determinados caracteres, por ejemplo,
blanco, coma, punto, 2 puntos en el caso del lenguaje escrito, mas, menos,
slash en el caso de matematicas o codigo.
De esta forma si recibe como identificador agua no se lo pone a paraguas.
En sensible a mayusculas y minusculas.
Los operadores se reciben como un texto y cada caracter es un operador.
Si el texto de operadores es vacio asume una lista estandar.
Si la tabla de etiquetas y links es vacia, no hace nada.",
HtmPutLinks);
////////////////////////////////////////////////////////////////////////////
```

Text HtmPutCharts()

```
////////////////////////////////////////////////////////////////////////////
Text HtmPutCharts(Text codTxt) // Codigo Html de entrada
////////////////////////////////////////////////////////////////////////////
{
  // Los guiones - son realmente _
  // <span class='CodRem'>//-(chartName)-</span>
  Text iniTag = "<span class='CodRem'>/"+"/_(";
  Text endTag = ")_"+"</span>";

  If(!TextFind(codTxt, iniTag), codTxt, // No hay graficos online
  {
    Text WriteLn("Code with online charts");
    Text iniImg = "<img src='";
    Text endImg =
      "' class='Tin' title='pulsando se cambia el tamaño' onclick="+
    Q("JavaScript:if(this.className=='Tin') { this.className='Mid'; } "+
                                        "else { this.className='Tin'; }")+">";

    Text iniRep = Replace(codTxt, iniTag, iniImg);
    Text endRep = Replace(iniRep, endTag, endImg);

    endRep
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Si hay graficos online inserta las instrucciones para que se vean.
Si no hay graficos online retorna el mismo codigo Html que recibe como
parametro.",
HtmPutCharts);
////////////////////////////////////////////////////////////////////////////
```

shi.tol de Dct.Writer

Funciones de sintaxis realzada (syntax highlight).

Declaraciones

Constantes

- `Text SHiRemOpn`
  Inicio tipico de un comentario de bloque.
- `Text SHiRemCls`
  Cierre tipico de un comentario de bloque.
- `Text SHiQuo`
  Comillas dobles.
- `Text SHiRemHtm`
  Span para un comentario.
- `Text SHiQu1Htm`
  Span para un texto de primer nivel.
- `Text SHiQu2Htm`
  Span para un subtexto de segundo nivel.
- `Text SHiTagHtm`
  Span para el inicio de una etiqueta.
- `Text SHiCodEmb`
  Span para el codigo embebido.
- `Text SHiEndHtm`
  Cierre de un span.

Funciones

- `Set SHiRemStrCodSet(Text txtInp, Text remOpn, Text remCls, Text remLin, Text strQu1, Text strQu2)`
  Retorna un conjunto de textos resultado de separar los comentarios de bloque, los comentarios de linea, los textos entrecomillados y el codigo.
- `Text SHiRemStrCodHtm(Text txtInp, Text remOpn, Text remCls, Text remLin, Text strQu1, Text strQu2)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo de entrada txtImp. El ReplaceTable() para transformar < y > no se puede aplicar desde el principio porque a la hora de realzar codigo Html son los tags delimitadores mas importantes, por eso hay que hacerlo trozo a trozo de codigo.

- `Text SHiPre(Text txtInp)`
  Retorna el codigo Html resultado de enmarcar txtImp entre las etiquetas <pre><code> y </code></pre>.
- `Text SHiRemStrCodPre(Text txtInp, Text remOpn, Text remCls, Text remLin, Text strQu1, Text strQu2)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo de txtImp enmarcado todo el resultado entre las etiquetas <pre><code> y </code></pre>.
- `Text SHiTolHtm(Text codInp)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo Tol de entrada que recibe como parametro pero sin enmarcar entre las etiquetas <pre><code> y </code> </pre>.
- `Text SHiTolPre(Text codInp)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo Tol de entrada que recibe como parametro.
- `Text SHiJscHtm(Text codInp)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo Javascript de entrada que recibe como parametro pero sin enmarcar entre las etiquetas <pre><code> y </code></pre>.
- `Text SHiJscPre(Text codInp)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo Javascript de entrada que recibe como parametro.
- `Text SHiSqlPre(Text codInp)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo Sql de entrada que recibe como parametro.
- `Text SHiGplPre(Text codInp)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo Gnuplot de entrada que recibe como parametro.
- `Text SHiCmdPre(Text codInp)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo de mandatos para el Cmd de Dos que recibe como parametro.
- `Text SHiCssPre(Text codInp)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo Cascade style sheet que recibe como parametro.
- `Text SHiXmlHtm(Text codInp)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo Xml que recibe como parametro pero sin enmarcar en las etiquetas <pre><code> y </code></pre> Emplea la version no recursiva TxtSplitBy2Fast(), aunque menos robusta que TxtSplitBy2Tag(), para evitar caidas en fichero Xml de cierta longitud.
- `Text SHiXmlPre(Text codInp)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo Xml que recibe como parametro.
- `Text SHiEmbHtm(Text codInp)`

Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo Html basico que recibe como parametro pero sin enmarcar en las etiquetas <pre><code> y </code></pre>.

- `Text SHiEmbPre(Text codInp)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo Xml que recibe como parametro.
- `Text SHiAscPre(Text codInp)`
  Retorna el codigo Html del contenido Ascii que recibe como parametro. Unicamente cambia los caracteres especiales con HtmAsc2Xml() y pone las etiquetas <pre><code> y </code></pre>
- `Text SHiImgSrc(Text filPth)`
  Retorna el codigo Html para visualizar el fichero grafico, cuya ruta recibe como parametro, de una forma simple.
- `Text SHiCompact(Text codHtm)`
  Retorna el codigo Html resultado de eliminar las lineas en blanco antes y despues del <pre><code> y </code></pre>.
- `Text SHiPreCode(Text filPth, Text filTxt)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo filTxt que se la leido, total o parcialmente, de fichero filPth. Recibe el nombre o ruta del fichero para saber su extension y de ahi el lenguaje de programacion.
- `Text SHiPreFile(Text filPth)`
  Retorna el codigo Html resultado de aplicar sintaxis realzada a todo el codigo del fichero que recibe como parametro.

Constantes

Text SHiRemOpn

```
////////////////////////////////////////////////////////////////////////
Text SHiRemOpn = "/"+"*";
////////////////////////////////////////////////////////////////////////
PutDescription("Inicio tipico de un comentario de bloque.", SHiRemOpn);
////////////////////////////////////////////////////////////////////////
```

Text SHiRemCls

```
////////////////////////////////////////////////////////////////////////
Text SHiRemCls = "*"+"/";
////////////////////////////////////////////////////////////////////////
PutDescription("Cierre tipico de un comentario de bloque.", SHiRemCls);
////////////////////////////////////////////////////////////////////////
```

Text SHiQuo

```
////////////////////////////////////////////////////////////////////////
Text SHiQuo = Char(34);
////////////////////////////////////////////////////////////////////////
PutDescription("Comillas dobles.", SHiQuo);
////////////////////////////////////////////////////////////////////////
```

Text SHiRemHtm

```
////////////////////////////////////////////////////////////////////////
Text SHiRemHtm = "<span class='CodRem'>";
////////////////////////////////////////////////////////////////////////
PutDescription("Span para un comentario.", SHiRemHtm);
////////////////////////////////////////////////////////////////////////
```

Text SHiQu1Htm

```
////////////////////////////////////////////////////////////////////////
Text SHiQu1Htm = "<span class='CodTxt'>";
////////////////////////////////////////////////////////////////////////
PutDescription("Span para un texto de primer nivel.", SHiQu1Htm);
////////////////////////////////////////////////////////////////////////
```

Text SHiQu2Htm

```
////////////////////////////////////////////////////////////////////////
Text SHiQu2Htm = "<span class='CodSub'>";
////////////////////////////////////////////////////////////////////////
PutDescription("Span para un subtexto de segundo nivel.", SHiQu2Htm);
////////////////////////////////////////////////////////////////////////
```

Text SHiTagHtm

```
////////////////////////////////////////////////////////////////////////
Text SHiTagHtm = "<span class='CodTag'>";
////////////////////////////////////////////////////////////////////////
PutDescription("Span para el inicio de una etiqueta.", SHiTagHtm);
////////////////////////////////////////////////////////////////////////
```

Text SHiCodEmb

```
////////////////////////////////////////////////////////////////////////
Text SHiCodEmb = "<span class='CodEmb'>";
////////////////////////////////////////////////////////////////////////
PutDescription("Span para el codigo embebido.", SHiCodEmb);
////////////////////////////////////////////////////////////////////////
```

Text SHiEndHtm

```
////////////////////////////////////////////////////////////////////////
Text SHiEndHtm = "</span>";
////////////////////////////////////////////////////////////////////////
PutDescription("Cierre de un span.", SHiEndHtm);
////////////////////////////////////////////////////////////////////////
```

Set SHiRemStrCodSet()

```
////////////////////////////////////////////////////////////////////////
Set SHiRemStrCodSet(Text txtInp, // Texto de entrada
```

```
                    Text remOpn, // Apertura de los comentarios de bloque
                    Text remCls, // Cierre de los comentarios de bloque
                    Text remLin, // Apertura de los comentarios de linea
                    Text strQu1, // Delimitador de textos alto nivel (ie "")
                    Text strQu2) // Delimitacor de textos bajo nivel (ie '')
////////////////////////////////////////////////////////////////////////////
{
  Set  setBlk = If(remOpn == "", [[txtInp]], // No hay comentarios de bloque
                   TxtSplitBy2Tag(txtInp, remOpn, remCls)); // Comenta bloque

  Set  setQu1 = BinGroup("<<", EvalSet(setBlk, Set(Text txtCod)
  {
    If(TxtBeginStrict(txtCod, remOpn), [[txtCod]], // Era comentario de bloque
       TxtSplitBy1Tag(txtCod, strQu1))   // Romper por textos entrecomillados
  })); // De conjunto de conjuntos a conjunto lineal

  Set  setQu2 = If(strQu2 == "", setQu1, // Si no se emplea
  {
    BinGroup("<<", EvalSet(setQu1, Set(Text txtCod)
    {
      If(TxtBeginStrict(txtCod, remOpn), [[txtCod]], // Era comentario bloque
      If(TxtBeginStrict(txtCod, strQu1), [[txtCod]], // Quote alto nivel
         TxtSplitBy1Tag(txtCod, strQu2)))  // Romper textos y entrecomillados
    }))  // De conjunto de conjuntos a conjunto lineal
  });

  Set  setLin = If(remLin == "", setQu2, // Si no se emplea
  {
    BinGroup("<<", EvalSet(setQu2, Set(Text txtCod)
    {
      If(TxtBeginStrict(txtCod, remOpn), [[txtCod]], // Era comentario bloque
      If(TxtBeginStrict(txtCod, strQu1), [[txtCod]], // Quote alto nivel
      If(TxtBeginStrict(txtCod, strQu2), [[txtCod]], // Quote bajo nivel
         TxtSplitBy2Tag(txtCod, remLin, "\n"))))  // Romper comentarios linea
    })) // De conjunto de conjuntos a conjunto lineal
  });

  setLin
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto de textos resultado de separar los comentarios de bloque,
los comentarios de linea, los textos entrecomillados y el codigo.",
SHiRemStrCodSet);
////////////////////////////////////////////////////////////////////////////
```

Text SHiRemStrCodHtm()

```
////////////////////////////////////////////////////////////////////////////
Text SHiRemStrCodHtm(Text txtInp, // Texto de entrada
                     Text remOpn, // Apertura de los comentarios de bloque
                     Text remCls, // Cierre de los comentarios de bloque
                     Text remLin, // Apertura de los comentarios de linea
                     Text strQu1, // Delimitador de textos alto nivel (ie "")
                     Text strQu2) // Delimitacor de textos bajo nivel (ie '')
////////////////////////////////////////////////////////////////////////////
{
  Set  setBlk = SHiRemStrCodSet(txtInp,remOpn,remCls,remLin,strQu1,strQu2);
  Set  setHtm = EvalSet(setBlk, Text(Text txtCod)
  {
    Case(
      TxtBeginStrict(txtCod, remOpn),
        SHiRemHtm + HtmAsc2Xml(txtCod) + SHiEndHtm,

      TxtBeginStrict(txtCod, strQu1),
        SHiQu1Htm + HtmAsc2Xml(txtCod) + SHiEndHtm,

      TxtBeginStrict(txtCod, strQu2),
        SHiQu2Htm + HtmAsc2Xml(txtCod) + SHiEndHtm,

      TxtBeginStrict(txtCod, remLin),
        SHiRemHtm + Replace(HtmAsc2Xml(txtCod), "\n", SHiEndHtm+"\n"),
```

```
      TRUE,
        HtmAsc2Xml(txtCod))
  });
  SetSum(setHtm)
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo de
entrada txtImp.
El ReplaceTable() para transformar < y > no se puede aplicar desde el
principio porque a la hora de realzar codigo Html son los tags delimitadores
mas importantes, por eso hay que hacerlo trozo a trozo de codigo.",
SHiRemStrCodHtm);
////////////////////////////////////////////////////////////////////////
```

Text SHiPre()

```
////////////////////////////////////////////////////////////////////////
Text SHiPre(Text txtInp) // Texto de entrada
////////////////////////////////////////////////////////////////////////
{ "\n<pre><code>" + txtInp + "</code></pre>\n" };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de enmarcar txtImp entre las etiquetas
<pre><code> y </code></pre>.",
SHiPre);
////////////////////////////////////////////////////////////////////////
```

Text SHiRemStrCodPre()

```
////////////////////////////////////////////////////////////////////////
Text SHiRemStrCodPre(Text txtInp, // Texto de entrada
                     Text remOpn, // Apertura de los comentarios de bloque
                     Text remCls, // Cierre de los comentarios de bloque
                     Text remLin, // Apertura de los comentarios de linea
                     Text strQu1, // Delimitador de textos alto nivel (ie "")
                     Text strQu2) // Delimitacor de textos bajo nivel (ie '')
////////////////////////////////////////////////////////////////////////
{ SHiPre(SHiRemStrCodHtm(txtInp,remOpn,remCls,remLin,strQu1,strQu2)) };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo de
txtImp enmarcado todo el resultado entre las etiquetas <pre><code> y
</code></pre>.",
SHiRemStrCodPre);
////////////////////////////////////////////////////////////////////////
```

Text SHiTolHtm()

```
////////////////////////////////////////////////////////////////////////
Text SHiTolHtm(Text codInp) // Codigo de entrada Tol
////////////////////////////////////////////////////////////////////////
{ SHiRemStrCodHtm(codInp, SHiRemOpn, SHiRemCls, "//", SHiQuo, "") };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
Tol de entrada que recibe como parametro pero sin enmarcar entre las
etiquetas <pre><code> y </code></pre>.",
SHiTolHtm);
////////////////////////////////////////////////////////////////////////
```

Text SHiTolPre()

```
////////////////////////////////////////////////////////////////////////
Text SHiTolPre(Text codInp) // Codigo de entrada Tol
////////////////////////////////////////////////////////////////////////
{ SHiPre(SHiTolHtm(codInp)) };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
Tol de entrada que recibe como parametro.",
SHiTolPre);
////////////////////////////////////////////////////////////////////////
```

Text SHiJscHtm()

```
////////////////////////////////////////////////////////////////////////
Text SHiJscHtm(Text codInp) // Codigo de entrada Javascript
////////////////////////////////////////////////////////////////////////
{ SHiRemStrCodHtm(codInp, SHiRemOpn, SHiRemCls, "//", SHiQuo, "'") };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
Javascript de entrada que recibe como parametro pero sin enmarcar entre las
etiquetas <pre><code> y </code></pre>.",
SHiJscHtm);
////////////////////////////////////////////////////////////////////////
```

Text SHiJscPre()

```
////////////////////////////////////////////////////////////////////////
Text SHiJscPre(Text codInp) // Codigo de entrada Javascript
////////////////////////////////////////////////////////////////////////
{ SHiPre(SHiJscHtm(codInp)) };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
Javascript de entrada que recibe como parametro.",
SHiJscPre);
////////////////////////////////////////////////////////////////////////
```

Text SHiSqlPre()

```
////////////////////////////////////////////////////////////////////////
Text SHiSqlPre(Text codInp) // Codigo de entrada Sql
////////////////////////////////////////////////////////////////////////
{ SHiRemStrCodPre(codInp, "", "", "--", "'", "") };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
Sql de entrada que recibe como parametro.",
SHiSqlPre);
////////////////////////////////////////////////////////////////////////
```

Text SHiGplPre()

```
////////////////////////////////////////////////////////////////////////
Text SHiGplPre(Text codInp) // Codigo de entrada Gnuplot
////////////////////////////////////////////////////////////////////////
{ SHiRemStrCodPre(codInp, "", "", "#", "'", "") };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
Gnuplot de entrada que recibe como parametro.",
```

```
SHiGplPre);
////////////////////////////////////////////////////////////////////////////
```

Text SHiCmdPre()

```
////////////////////////////////////////////////////////////////////////////
Text SHiCmdPre(Text codInp) // Codigo de entrada de mandatos Dos
////////////////////////////////////////////////////////////////////////////
{ SHiRemStrCodPre(codInp, "", "", "::", SHiQuo, "") };
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
de mandatos para el Cmd de Dos que recibe como parametro.",
SHiCmdPre);
////////////////////////////////////////////////////////////////////////////
```

Text SHiCssPre()

```
////////////////////////////////////////////////////////////////////////////
Text SHiCssPre(Text codInp) // Codigo de entrada Cascade style sheet
////////////////////////////////////////////////////////////////////////////
{ SHiRemStrCodPre(codInp, SHiRemOpn, SHiRemCls, "", SHiQuo, "'") };
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
Cascade style sheet que recibe como parametro.",
SHiCssPre);
////////////////////////////////////////////////////////////////////////////
```

Text SHiXmlHtm()

```
////////////////////////////////////////////////////////////////////////////
Text SHiXmlHtm(Text codInp) // Codigo Xml
////////////////////////////////////////////////////////////////////////////
{
  Set  setBlk = TxtSplitBy2Tag(codInp, "<!--", "-->"); // Romper x comments

  Set  setTag = BinGroup("<<", EvalSet(setBlk, Set(Text txtCod)
  {
    If(TxtBeginStrict(txtCod, "<!--"), [[txtCod]], // Era comentario
       TxtSplitBy2Fast(txtCod, "<", ">")) // Romper x etiquetas, no recursivo
  })); // De conjunto de conjuntos a conjunto lineal

  Set  setHtm = EvalSet(setTag, Text(Text txtCod)
  {
    Case(
      TxtBeginStrict(txtCod, "<!--"), // Comentario
        SHiRemHtm + HtmAsc2Xml(txtCod) + SHiEndHtm,

      Or(TxtBeginStrict(txtCod, "<"), TxtEndStrict(Compact(txtCod), ">")),
        SHiTagHtm + SHiJscHtm(txtCod) + SHiEndHtm, // Etiqueta

      TRUE,
        HtmAsc2Xml(txtCod))
  });
  SetSum(setHtm)
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
Xml que recibe como parametro pero sin enmarcar en las etiquetas
<pre><code> y </code></pre>
Emplea la version no recursiva TxtSplitBy2Fast(), aunque menos robusta que
TxtSplitBy2Tag(), para evitar caidas en fichero Xml de cierta longitud.",
SHiXmlHtm);
////////////////////////////////////////////////////////////////////////////
```

Text SHiXmlPre()

```
////////////////////////////////////////////////////////////////////////
Text SHiXmlPre(Text codInp) // Codigo Xml
////////////////////////////////////////////////////////////////////////
{ SHiPre(SHiXmlHtm(codInp)) };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
Xml que recibe como parametro.",
SHiXmlPre);
////////////////////////////////////////////////////////////////////////
```

Text SHiEmbHtm()

```
////////////////////////////////////////////////////////////////////////
Text SHiEmbHtm(Text codInp) // Codigo Html con Javscript y Tol embebido
////////////////////////////////////////////////////////////////////////
{
  Set  setTol = TxtSplitBy2Tag(codInp, "<"+"{", "}"+">"); // Tol embebido

  Set  setJsc = BinGroup("<<", EvalSet(setTol, Set(Text txtCod)
  {
    If(TxtBeginStrict(txtCod, "<"+"{"), [[txtCod]], // Es Tol embebido
       TxtSplitBy2Tag(txtCod, "<script", "</script>"))   // Javascript
  })); // De conjunto de conjuntos a conjunto lineal


  Set  setHtm = EvalSet(setJsc, Text(Text txtCod)
  {
    Case(
      TxtBeginStrict(txtCod, "<"+"{"), // Tol embebido
        SHiCodEmb + SHiTolHtm(txtCod) + SHiEndHtm,

      TxtBeginStrict(txtCod, "<script"), // Javascript embebido
        SHiCodEmb + SHiJscHtm(txtCod) + SHiEndHtm,

      TRUE, // Resto de codigo con tags Xml / Html
        SHiXmlHtm(txtCod))
  });
  SetSum(setHtm)
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
Html basico que recibe como parametro pero sin enmarcar en las etiquetas
<pre><code> y </code></pre>.",
SHiEmbHtm);
////////////////////////////////////////////////////////////////////////
```

Text SHiEmbPre()

```
////////////////////////////////////////////////////////////////////////
Text SHiEmbPre(Text codInp) // // Codigo Html con Javscript y Tol embebido
////////////////////////////////////////////////////////////////////////
{ SHiPre(SHiEmbHtm(codInp)) };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
Xml que recibe como parametro.",
SHiEmbPre);
////////////////////////////////////////////////////////////////////////
```

Text SHiAscPre()

```
////////////////////////////////////////////////////////////////////////
Text SHiAscPre(Text codInp) // // Fichero Ascii
////////////////////////////////////////////////////////////////////////
{ SHiPre(HtmAsc2Xml(codInp)) };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html del contenido Ascii que recibe como parametro.
Unicamente cambia los caracteres especiales con HtmAsc2Xml() y pone las
etiquetas <pre><code> y </code></pre>",
SHiAscPre);
////////////////////////////////////////////////////////////////////////
```

Text SHiImgSrc()

```
////////////////////////////////////////////////////////////////////////
Text SHiImgSrc(Text  filPth) // Ruta al fichero grafico
////////////////////////////////////////////////////////////////////////
{ "\n<img src='"+filPth+"' class='Mid'>\n" };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html para visualizar el fichero grafico, cuya ruta recibe
como parametro, de una forma simple.",
SHiImgSrc);
////////////////////////////////////////////////////////////////////////
```

Text SHiCompact()

```
////////////////////////////////////////////////////////////////////////
Text SHiCompact(Text codHtm) // // Codigo Html ya autogenerado
////////////////////////////////////////////////////////////////////////
{
  ReplaceTable(codHtm, [[ [["<pre><code> ",     "<pre><code>"]],
                          [["<pre><code>\n",    "<pre><code>"]],
                          [[" </code></pre>",   "</code></pre>"]],
                          [["\n</code></pre>", "</code></pre>"]] ]])
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de eliminar las lineas en blanco antes y
despues del <pre><code> y </code></pre>.",
SHiCompact);
////////////////////////////////////////////////////////////////////////
```

Text SHiPreCode()

```
////////////////////////////////////////////////////////////////////////
Text SHiPreCode(Text filPth, // Nombre o ruta del fichero para la extension
                Text filTxt) // Codigo ya leido del fichero, todo o parte
////////////////////////////////////////////////////////////////////////
{
  Text filExt = ToLower(GetFileExtension(filPth));
  Text shiCod = Case(
    filExt == "tol",  SHiTolPre(filTxt), // Time Oriented Language
    filExt == "bst",  SHiTolPre(filTxt), // Fichero Bst de Tol
                                         // Bdt como Ascii por velocidad

    filExt == "bat",  SHiCmdPre(filTxt), // Command bat
    filExt == "css",  SHiCssPre(filTxt), // Cascade style sheet
    filExt == "gpl",  SHiGplPre(filTxt), // Gnuplot
    filExt == "js",   SHiJscPre(filTxt), // Javascript
    filExt == "sql",  SHiSqlPre(filTxt), // Sql
    filExt == "xml",  SHiXmlPre(filTxt), // Xml
```

```
    filExt == "htm",  SHiEmbPre(filTxt), // Htm
    filExt == "html", SHiEmbPre(filTxt), // Html
    filExt == "age",  SHiEmbPre(filTxt), // Html

    filExt == "gif",  SHiImgSrc(filPth), // Grafico Gif,  img src path
    filExt == "jpg",  SHiImgSrc(filPth), // Grafico Jpeg, img src path
    filExt == "png",  SHiImgSrc(filPth), // Grafico Png,  img src path

    TRUE,             SHiAscPre(filTxt)  // Ascii, limpiar y con pre y code
  );
  SHiCompact(shiCod)
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada al codigo
filTxt que se la leido, total o parcialmente, de fichero filPth.
Recibe el nombre o ruta del fichero para saber su extension y de ahi el
lenguaje de programacion.",
SHiPreCode);
////////////////////////////////////////////////////////////////////////////
```

Text SHiPreFile()

```
////////////////////////////////////////////////////////////////////////////
Text SHiPreFile(Text filPth) // Ruta del fichero de entrada
////////////////////////////////////////////////////////////////////////////
{ SHiPreCode(filPth, ReadFile(filPth)) };
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el codigo Html resultado de aplicar sintaxis realzada a todo el
codigo del fichero que recibe como parametro.",
SHiPreFile);
////////////////////////////////////////////////////////////////////////////
```

dtr.tol de Dct.Writer

Funciones de arboles de estructuras de directorios para dar soporte a la generacion de la documentacion de programas en lenguaje Tol y en otros lenguajes de programacion.

Declaraciones

Constantes

- `Text DtrMrk`
  Marca interna para identificar la raiz del arbol y del modulos principal.
- `Text DtrOut`
  Marca interna para identificar los ficheros de salida.

Funciones

- `Text DTrAddPath(Text inpPth, Text subPth)`
  Retorna el resultado de concatenar 2 trozos de una ruta de un fichero.
- `Text DTrCompact(Real delMrk, Text txtDes)`
  Retorna una linea de documentación compactada y sin la marca de principal o de fichero de salida si se le pide con el argumento delMrk.
- `Text DTrRootFile(Set inpTre)`
  Retorna el fichero principal de un arbol de ficheros.
- `Text DTrText(Set inpTre, Text inpPth, Real padLft)`
  Retorna en texto plano con saltos de linea e indentaciones la estructura del arbol a documentar. Es una funcion auxiliar, permite comprobar visualmente el contenido de dicho arbol sin uso para la documentacion.
- `Text DTrHtml(Set inpTre, Text prgNam, Real padLft, Text actFil)`
  Retorna en html el contenido del arbol, mediante los tags ul y li ademas de p, div y span, con el Css adecuado la visualizacion que legible. Pone si recibe actFil el nombre de uno de los ficheros del arbol lo destaca como fichero actual, poniendo al resto como simples hojas. Si no recibe ningun fichero pone como fichero actual el principal del arbol que ha de estar marcado con la etiqueta DTrMrk al final de su linea de documentacion. A todos los ficheros les incluye un enlace a su pagina de documentacion.
- `Set DtrFileSet(Set inpTre, Text inpPth)`
  Retorna una tabla con el conjunto de los ficheros del arbol, hojas terminales, que hay que procesar, con sus descripciones compactadas y con un primer campo que identifica el fichero principal, Rot, porque pasa a ser la raiz del web de paginas del proyecto), mientras que el resto de ficheros son hoja, Lea. La tabla tiene 6 columnas: 1) identificador de si es el principal, 2) ruta del fichero incluyendo su nombre 3) nombre del fichero, 4) la descripcion compactada y sin marcas de fichero principal, 5) la etiqueta inicial de corte, si no procede, la tira vacia, y 6) la etiqueta final de corte, si no procede, la tira vacia. Los 2 ultimos campos sirven para cuando solo se desea documentar un trozo de un fichero entre 2 etiquetas, si son vacias, se asume el fichero completo. Esta programacion es un ejemplo de como se transforma en lineal una estructura de arbol, si bien en este caso solo de sus hojas. Para localizar los ficheros utiliza un inicio de camino inpPth. Si el fichero a documentar no existe entonces introduce un mensaje de error dentro del campo de documentacion. Notese que

el uso se facilita y el nombre del directorio principal y de como se denomina al programa coinciden. El fichero puede no existir, por ejemplo, cuando se ponen ficheros genericos de ejemplo de muchas paginas web que se complementan con imagenes y no con el codigo interno de esas paginas.

- `Set DtrFileExc(Set filTab)`
  Retorna una tabla con el conjunto de informacion de ficheros de entrada si todo cumplen los criterios de inclusion y Empty en otro caso.

Constantes

Text DtrMrk

```
////////////////////////////////////////////////////////////////////////////
Text DtrMrk = "[+]";
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Marca interna para identificar la raiz del arbol y del modulos principal.",
DtrMrk);
////////////////////////////////////////////////////////////////////////////
```

Text DtrOut

```
////////////////////////////////////////////////////////////////////////////
Text DtrOut = "[>]";
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Marca interna para identificar los ficheros de salida.",
DtrOut);
////////////////////////////////////////////////////////////////////////////
```

Text DTrAddPath()

```
////////////////////////////////////////////////////////////////////////////
Text DTrAddPath(Text inpPth, // Ruta superior de entrada
                Text subPth) // Subruta que se concatena
////////////////////////////////////////////////////////////////////////////
{ TxtCat(inpPth, "/", subPth) };
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el resultado de concatenar 2 trozos de una ruta de un fichero.",
DTrAddPath);
////////////////////////////////////////////////////////////////////////////
```

Text DTrCompact()

```
////////////////////////////////////////////////////////////////////////////
Text DTrCompact(Real delMrk, // Si cierto elimina las marcas
                Text txtDes) // Texto de descripcion
////////////////////////////////////////////////////////////////////////////
{
  Compact(If(delMrk, ReplaceTable(txtDes, [[ [[DtrMrk, ""]],
                                             [[DtrOut, ""]] ]]), txtDes))
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna una linea de documentación compactada y sin la marca de principal
o de fichero de salida si se le pide con el argumento delMrk.",
```

```
DTrCompact);
////////////////////////////////////////////////////////////////////////
```

Text DTrRootFile()

```
////////////////////////////////////////////////////////////////////////
Text DTrRootFile(Set inpTre) // Arbol de entrada
////////////////////////////////////////////////////////////////////////
{
  Set filTab = DtrFileSet(inpTre, ""); // No necesita el path completo
  Set rotSet = Select(filTab, Real(Set filSet) { filSet[1]=="Rot" });
  rotSet[1][3]
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el fichero principal de un arbol de ficheros.", DTrRootFile);
////////////////////////////////////////////////////////////////////////
```

Text DTrText()

```
////////////////////////////////////////////////////////////////////////
Text DTrText(Set  inpTre, // Arbol de entrada
             Text inpPth, // Directorio previo
             Real padLft) // Numero de blancos a la izquierda
////////////////////////////////////////////////////////////////////////
{
  Text whiLft = Repeat("  ",padLft); // Blancos a la izquierda
  If(NE(Card(inpTre), 3), // Las hojas son de 2 o 4
  { // Hoja
    Real leaAct = TextEndAt(inpTre[2], DtrMrk);
    Real leaOut = TextEndAt(inpTre[2], DtrOut);
    Text leaDes = DTrCompact(Or(leaAct,leaOut), inpTre[2]);
    Text leaTyp = Case(leaAct, "Act",
                       leaOut, "Out",
                       TRUE,   "Lea");
    whiLft+leaTyp+": "+DTrAddPath(inpPth,inpTre[1])+"|"+leaDes+"\n"
  },
  { // Nodo
    Text newPth = DTrAddPath(inpPth,inpTre[1]);
    Real nodRot = TextEndAt(inpTre[2], DtrMrk);
    Text nodDes = DTrCompact(nodRot, inpTre[2]);
    Text nodTyp = If(nodRot, "Rot", "Nod");

    whiLft+nodTyp+": "+newPth+"|"+nodDes+"\n"+
    SetSum(EvalSet(inpTre[3], Text(Set subNod)
                   { DTrText(subNod, newPth, padLft+1) }))
  })
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna en texto plano con saltos de linea e indentaciones la estructura del
arbol a documentar.
Es una funcion auxiliar, permite comprobar visualmente el contenido de dicho
arbol sin uso para la documentacion.",
DTrText);
////////////////////////////////////////////////////////////////////////
```

Text DTrHtml()

```
////////////////////////////////////////////////////////////////////////
Text DTrHtml(Set  inpTre, // Arbol de entrada
             Text prgNam, // Directorio de proyecto
             Real padLft, // Numero de blancos a la izquierda
             Text actFil) // Fichero actual
////////////////////////////////////////////////////////////////////////
```

```
{
  Text whiLft = Repeat(" ",padLft); // Blancos a la izquierda
  If(NE(Card(inpTre), 3), // Las hojas son de 2 o 4
  { // Hoja
    Text ahrIni = DctLink(prgNam, inpTre[1], "", "A"); // A href y Url

    Real endMrk = TextEndAt(inpTre[2], DtrMrk);
    Real leaOut = TextEndAt(inpTre[2], DtrOut);
    Text leaDes = DTrCompact(Or(endMrk, leaOut), inpTre[2]);

    Text leaTyp =Case(
     actFil==inpTre[1], "Act",  // Es el actual, azul
     endMrk,            "Rot",  // Es el principal, rojo
     leaOut,            "Out",  // Es un fichero de salida, verde
     TRUE,              "Lea"); // Es normal, gris

    Text arrTxt = If(leaOut, "<b>&rarr;</b>", "<b>&larr;</b>");

    whiLft+"<li>"+arrTxt+ahrIni+"<span class="+Q(leaTyp)+">"+inpTre[1]+
                               "</span></a> "+leaDes+"</li>\n"
  },
  { // Nodo: raiz o normal
    If(TextEndAt(inpTre[2], DtrMrk),
    { // Raiz
      Text nodDes = DTrCompact(TRUE, inpTre[2]);
      Text ahrIni = DctLink(prgNam, DTrRootFile(inpTre), "", "A");

      whiLft+"<div class='Tre'>\n"+
      whiLft+"<p>"+ahrIni+"<span class='Rot'>"+inpTre[1]+
                          "</span></a> "+nodDes+"</p>\n"+
      whiLft+"  <ul>\n"+
      SetSum(EvalSet(inpTre[3], Text(Set subNod)
                     { DTrHtml(subNod, prgNam, padLft+4, actFil) })) +
      whiLft+"  </ul>\n"+
      whiLft+"</div>\n"
    },
    { // Nodo normal
      Text nodDes = DTrCompact(FALSE, inpTre[2]);

      whiLft+"<li><span class='Nod'>"+inpTre[1]+"</span> "+nodDes+"\n" +
      whiLft+"  <ul>\n" +
      SetSum(EvalSet(inpTre[3], Text(Set subNod)
                     { DTrHtml(subNod, prgNam, padLft+4, actFil) })) +
      whiLft+"  </ul>\n" +
      whiLft+"</li>\n"
    })
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna en html el contenido del arbol, mediante los tags ul y li ademas de
p, div y span, con el Css adecuado la visualizacion que legible.
Pone si recibe actFil el nombre de uno de los ficheros del arbol lo destaca
como fichero actual, poniendo al resto como simples hojas.
Si no recibe ningun fichero pone como fichero actual el principal del arbol
que ha de estar marcado con la etiqueta DTrMrk al final de su linea de
documentacion.
A todos los ficheros les incluye un enlace a su pagina de documentacion.",
DTrHtml);
////////////////////////////////////////////////////////////////////////////
```

Set DtrFileSet()

```
////////////////////////////////////////////////////////////////////////////
Set DtrFileSet(Set  inpTre, // Arbol de entrada
               Text inpPth) // Directorio previo
////////////////////////////////////////////////////////////////////////////
{
  Real inpCrd = Card(inpTre);
  If(NE(Card(inpTre), 3), // Las hojas son de 2 o 4
  { // Hoja
    Real leaRot = TextEndAt(inpTre[2], DtrMrk);
```

```
    Real leaOut = TextEndAt(inpTre[2], DtrOut);
    Text leaDes = DTrCompact(Or(leaRot,leaOut), inpTre[2]);

    Text leaTyp = If(leaRot, "Rot", "Lea");
    Text filPth = DTrAddPath(inpPth,inpTre[1]);
    Text iniTag = If(EQ(inpCrd, 4), inpTre[3], "");
    Text endTag = If(EQ(inpCrd, 4), inpTre[4], "");

    [[ SetOfText(leaTyp, filPth, inpTre[1], leaDes, iniTag, endTag) ]]
  },
  { // Nodo
    Text newPth = DTrAddPath(inpPth,inpTre[1]);

    BinGroup("<<", EvalSet(inpTre[3], Set(Set subNod)
                           { DtrFileSet(subNod, newPth) }))
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna una tabla con el conjunto de los ficheros del arbol, hojas
terminales, que hay que procesar, con sus descripciones compactadas y
con un primer campo que identifica el fichero principal, Rot, porque
pasa a ser la raiz del web de paginas del proyecto), mientras que el
resto de ficheros son hoja, Lea.
La tabla tiene 6 columnas:
1) identificador de si es el principal,
2) ruta del fichero incluyendo su nombre
3) nombre del fichero,
4) la descripcion compactada y sin marcas de fichero principal,
5) la etiqueta inicial de corte, si no procede, la tira vacia, y
6) la etiqueta final de corte, si no procede, la tira vacia.
Los 2 ultimos campos sirven para cuando solo se desea documentar un trozo de
un fichero entre 2 etiquetas, si son vacias, se asume el fichero completo.
Esta programacion es un ejemplo de como se transforma en lineal una estructura
de arbol, si bien en este caso solo de sus hojas.
Para localizar los ficheros utiliza un inicio de camino inpPth.
Si el fichero a documentar no existe entonces introduce un mensaje de error
dentro del campo de documentacion.
Notese que el uso se facilita y el nombre del directorio principal y de como
se denomina al programa coinciden.
El fichero puede no existir, por ejemplo, cuando se ponen ficheros genericos
de ejemplo de muchas paginas web que se complementan con imagenes y no con
el codigo interno de esas paginas.",
DtrFileSet);
////////////////////////////////////////////////////////////////////////////
```

Set DtrFileExc()

```
////////////////////////////////////////////////////////////////////////////
Set DtrFileExc(Set  filTab) // Tabla de ficheros de entrada
////////////////////////////////////////////////////////////////////////////
{
  Set selTab = Select(filTab, Real(Set filSet)
  {
    If(TextFind   (filSet[3], DctExc), FALSE, // Exclusion en el nombre
    If(FilTextFind(filSet[3], DctExc), FALSE, // Exclusion en el contenido
                                       TRUE))
  });
  Real numExc = Card(filTab)-Card(selTab);
  If(!numExc, selTab,
    { Text WriteLn("Exclusions: "+F(numExc)); Empty })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna una tabla con el conjunto de informacion de ficheros de entrada si
todo cumplen los criterios de inclusion y Empty en otro caso.",
DtrFileExc);
////////////////////////////////////////////////////////////////////////////
```

dct.tol de Dct.Writer

Funciones de docuentacion de Tol a una agenda de post.

Declaraciones

Constantes

- `Text DctTdy`
  Fecha del dia que se genera la documentacion.
- `Real DctDesLen`
  Numero maximo de caracteres de una descripcion.
- `Text DctUndLinHtm`
  Separador de underscores en Html.
- `Text DctUndLinAsc`
  Separador de underscores en Ascii.
- `Text DctTolSepBas`
  Separador basico de codigo Tol.
- `Text DctTolSepLow`
  Separador de codigo Tol de bajo nivel.
- `Text DctTolSepTop`
  Separador de codigo Tol de alto nivel.
- `Text DctBr2`
  Para forzar separaciones en comentarios.
- `Text DctPstSee`
  Semilla de un post.
- `Text DctPstSep`
  Separador entre posts.

Funciones

- `Set DctGetInterface(Text tolCod)`
  Retorna, para una variable o una funcion, un conjunto de textos formado por [clase function|variable, tipo de retorno, nombre, argumentos, descripcion], en el caso de variables los argumentos son la tira vacia. En caso de error retorna Empty.
- `Set DctGetHeader(Text codHea)`
  Retorna un conjunto con los 4 atributos de la cabecera de un fichero que son el nombre del fichero, el autor, sus clases y su proposito.
- `Set DctGetHeaderRobust(Text codHea, Text filNam, Text filPur)`
  Retorna un conjunto con los 4 atributos de la cabecera de un fichero que son el nombre del fichero, el autor, sus clases y su proposito.
- `Set DctTolSplit(Text codInp)`

Retorna un conjunto de secciones (FILE, CONSTANTS, FUNCTIONS, etc.) resultado de romper el codigo Tol que recibe como parametro por secciones. Cada seccion esta formada por un conjunto con su nombre y su conjunto de declaraciones, salvo: - la primera que contiene la información general de la cabecera y - las secciones de codigo en bruto que retorna un conjunto formado por el texto nombre de la foncion y un texto (no un conjunto) que es el codigo en bruto formateado.

- `Real DctRawCode(Set dctSec)`
  Retorna cierto si el segundo componente no existe o si existe no es un conjunto de declaraciones de codigo. Lo natural en caso de codigo bruto es que exista pero sea un texto con dicho codigo en bruto.
- `Text DctLink(Text prgNam, Text filNam, Text pstSec, Text ctrRet)`
  Retorna: - si ctrRet=F el fichero de salida Html, salvo los Pdf que son Pdf, - si ctrRet=L el enlace global al programa, fichero y seccion y - si ctrRet=A el tag a href al enlace global. Si no hay fichero filNam asume la pagina principal index, lo que no tendria que suceder. Si no hay seccion no añade el #seccion.
- `Text DctPstReplace(Text prgNam, Text filNam, Text pstSec, Text addCla, Text pstSta, Text pstAut, Text pstDes, Text pstHtm)`
  Retorna por reemplazamiento un post con todos sus campos.
- `Text DctSecReplace(Text pstSec)`
  Retorna los nombres en castellano de los diferentes tipos de secciones.
- `Text DctFileLanguage(Text filNam, Real ctrLar)`
  Retorna el nombre del lenguaje de programación en formato corto o largo dependiendo de ctrLar.
- `Set DctDeclare(Text prgNam, Text filNam, Text decCat, Text decTyp, Text decNam, Text decArg, Text ascDes)`
  Para una declaracion Tol retorna un set de 5 elementos formado por: a) clases: Variable/Funcion, tipo Tol y nombre b) declaracion: tipo nombre(argumentos) c) descripcion: tipo nombre(argumentos) y proposito d) codigo para un li con tipo nombre(argumentos) y proposito y e) a href link para que el codigo de otras funciones le enlacen.
- `Real DctAppendPost(Text outFil, Text prgNam, Text filNam, Text pstSec, Text addCla, Text pstSta, Text pstAut, Text pstDes, Text pstHtm)`
  Añade un post con todos sus campos al fichero de agenda llamado outFil. La cabecera no lleva separador, el resto es separador mas contenido.
- `Real DctWriteHeader(Text prgNam, Text filNam, Set dctCab, Text agePth, Text levCla)`
  Inicia la agenda escribiendo el post de la cabecera de un programa.
- `Real DctWriteTree(Text prgNam, Text filNam, Set dctCab, Text agePth, Set inpTre, Text levCla)`
  Escribe el post del arbol de navegacion de un programa con el fichero en curso activo.
- `Real DctWriteDisplay(Text prgNam, Text filNam, Set dctCab, Text agePth)`
  Escribe el post de la presentación realizada a mano de un fichero filNam de un programa prgNam.

- `Real DctWriteIndex(Text prgNam, Text filNam, Set dctSHi, Text agePth, Text funDmp)`
  Escribe el post del indice de declaraciones de variables y funciones de un fichero filNam de un programa prgNam. Es un indice con enlaces al resto de declaraciones, pero se le titula como declaraciones pues no indexa otras secciones.
- `Real DctWriteCode(Text prgNam, Text filNam, Set dctSHi, Text agePth, Set lnkTab)`
  Escribe todos los posts de secciones de codigo (constantes, funciones, etc.) y todos los post de cada una de las estructuras, constantes, funciones, etc. de un fichero filNam de un programa prgNam.
- `Real DctWriteSource(Text prgNam, Text filNam, Set dctCab, Text agePth, Text codTxt, Set lnkTab, Text codDmp)`
  Escribe el post de la presentación realizada a mano de un fichero filNam de un programa prgNam.
- `Real DctMake(Text dctPre, Set dctTre)`
  Proceso principal de documentacion de todo el arbol de fichero de entrada.
- `Set DctOcurrences(Text funDmp, Text codDmp, Real ctrOcc)`
  Retorna una tabla de funciones y numero de occurencias para aquellas funciones que no superan ctrOcc, como efecto latelar las visualiza.
- `Set DctLinkFileDeclaration(Text prgNam, Text filNam, Set dctSHi)`
  Retorna una tabla de pares nombre de variable o funcion y su enlace de un fichero Tol de un programa.
- `Set DctLinkAllDeclaration(Set filTab)`
  Retorna una tabla de pares nombre de variable o funcion y su enlace de todos los ficheros Tol de un programa.

Constantes

Text DctTdy

```
////////////////////////////////////////////////////////////////////////////
Text DctTdy = FormatDate(Today, "%c%Y/%m/%d");
////////////////////////////////////////////////////////////////////////////
PutDescription("Fecha del dia que se genera la documentacion.", DctTdy);
////////////////////////////////////////////////////////////////////////////
```

Real DctDesLen

```
////////////////////////////////////////////////////////////////////////////
Real DctDesLen = 350;
////////////////////////////////////////////////////////////////////////////
PutDescription("Numero maximo de caracteres de una descripcion.", DctDesLen);
////////////////////////////////////////////////////////////////////////////
```

Text DctUndLinHtm

```
////////////////////////////////////////////////////////////////////////////
Text DctUndLinHtm = Repeat("&#95;",78);
////////////////////////////////////////////////////////////////////////////
PutDescription("Separador de underscores en Html.", DctUndLinHtm);
////////////////////////////////////////////////////////////////////////////
```

Text DctUndLinAsc

```
////////////////////////////////////////////////////////////////////////////
Text DctUndLinAsc = Repeat("_",78);
////////////////////////////////////////////////////////////////////////////
PutDescription("Separador de underscores en Ascii.", DctUndLinAsc);
////////////////////////////////////////////////////////////////////////////
```

Text DctTolSepBas

```
////////////////////////////////////////////////////////////////////////////
Text DctTolSepBas = "\n"+Repeat("/",78)+"\n";
////////////////////////////////////////////////////////////////////////////
PutDescription("Separador basico de codigo Tol.", DctTolSepBas);
////////////////////////////////////////////////////////////////////////////
```

Text DctTolSepLow

```
////////////////////////////////////////////////////////////////////////////
Text DctTolSepLow = "\n"+DctTolSepBas;
////////////////////////////////////////////////////////////////////////////
PutDescription("Separador de codigo Tol de bajo nivel.", DctTolSepLow);
////////////////////////////////////////////////////////////////////////////
```

Text DctTolSepTop

```
////////////////////////////////////////////////////////////////////////////
Text DctTolSepTop = "\n"+DctTolSepLow+"//  ";
////////////////////////////////////////////////////////////////////////////
PutDescription("Separador de codigo Tol de alto nivel.", DctTolSepTop);
////////////////////////////////////////////////////////////////////////////
```

Text DctBr2

```
////////////////////////////////////////////////////////////////////////////
Text DctBr2 = Char(12); // Form feed
////////////////////////////////////////////////////////////////////////////
PutDescription("Para forzar separaciones en comentarios.", DctTolSepTop);
////////////////////////////////////////////////////////////////////////////
```

Text DctPstSee

```
////////////////////////////////////////////////////////////////////////////
Text DctPstSee = "
```

```
<Pst.Cla> _CLA_
<Pst.Sta> B
<Pst.Dte> _DTE_
<Pst.Aut> _AUT_
<Pst.Pth> web/dct_writer/dct_tol.html
<Pst.Lnk> ../dct_writer/dct_tol.html#Dct_Writer_Text_DctPstSee
<Pst.Glo> _GLO_
<Pst.Loc> Text DctPstSee
<Pst.Des> _DES_
<Pst.Htm>
_HTM_
<Pst.End>
";
////////////////////////////////////////////////////////////////////////////
PutDescription("Semilla de un post.", DctTolSepTop);
////////////////////////////////////////////////////////////////////////////
```

Text DctPstSep

```
////////////////////////////////////////////////////////////////////////////
Text DctPstSep = Repeat("_",78)+"\n";
////////////////////////////////////////////////////////////////////////////
PutDescription("Separador entre posts.", DctPstSep);
////////////////////////////////////////////////////////////////////////////
```

Set DctGetInterface()

```
////////////////////////////////////////////////////////////////////////////
Set DctGetInterface(Text tolCod)
////////////////////////////////////////////////////////////////////////////
{
  Text clsQuo(Text desTxt) { Replace(desTxt, Char(34), "") };

  Set  trePar = Parse(tolCod)[3]; // Acesso al arbol del parse

  Case(
    trePar[1][1] == "=";   // Si es variable
    {
      Text varCla = "variable";
      Text varTyp = trePar[1][3][1][1];          // Tipo
      Text varNam = trePar[1][3][1][3][1][1];    // Nombre
      Text varDeQ = trePar[2][3][1][1];          // Descripcion con quote
      Text varDes = clsQuo(varDeQ);              // Descripcion sin quote
      Text argTxt = "";                          // Argumentos no hay

      SetOfText(varCla, varTyp, varNam, argTxt, varDes)
    },
    trePar[1][1] == "#F#"; // Si es funcion
    {
      Text funCla = "function";
      Text funTyp = trePar[1][3][1][1];          // Tipo
      Text funNam = trePar[1][3][1][3][1][1];    // Nombre
      Text funDeQ = trePar[2][3][1][1];          // Descripcion con quote
      Text funDes = clsQuo(funDeQ);              // Descripcion sin quote

      Set  argSet = trePar[1][3][1][3][1][3];    // Conjunto de argumentos
      Real argNum = Card(argSet);                // Numero de argumentos

      Set  argCic = For(1, argNum, Text(Real argPos)
      {
        Set  argInf = argSet[argPos]; // Informacion del argumento
        Text argTyp = argInf[1];
        Text argNam = argInf[3][1][1];

        If(EQ(argPos, 1), "", ", ")+ argTyp+" "+argNam
      });
      Text argTxt = "(" + SetSum(argCic) + ")";
```

```
      SetOfText(funCla, funTyp, funNam, argTxt, funDes)
    },
    TRUE, Empty) // Ni variable ni funcion
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna, para una variable o una funcion, un conjunto de textos formado por
[clase function|variable, tipo de retorno, nombre, argumentos, descripcion],
en el caso de variables los argumentos son la tira vacia.
En caso de error retorna Empty.",
DctGetInterface);
////////////////////////////////////////////////////////////////////////
```

Set DctGetHeader()

```
////////////////////////////////////////////////////////////////////////
Set DctGetHeader(Text codHea) // Trozo de codigo o todo para extraer atributos
////////////////////////////////////////////////////////////////////////
{
  Text repLar = ReplaceTable(codHea, [[ // Reemplazar los largos
    [["-- ///", "//////"]],                                    // Sql
    [[""/*////", "//////"]], [[ "////*/", "//////"]],          // Css
    [["/*-///", "//////"]], [[ "///-*/", "//////"]],          // Css
    [["# ////", "//////"]], [[ "#/////", "//////"]],          // Glp
    [[":: ///", "//////"]], [[ "::////", "//////"]],          // Bat
    [["<!-- /", "//////"]], [[ "// -->", "//////"]] ]], 1);  // Htm

  Text repSor = ReplaceTable(repLar, [[ // Reemplazar los cortos
    [["--",  "//"]],  // Sql
    [["# ",  "//"]],  // Glp
    [["::",  "//"]],  // Bat
    [[" : ", ": "]] ]], 4); // Como maximo 4 blancos antes de :

  Text filHea = repSor+"\n"; // Asegurarse el ultimo salto de linea

  Text filNam = TxtBetween2Tag(filHea, "FILE: ",    "\n", TRUE);
  Text autIde = TxtBetween2Tag(filHea, "AUTHOR: ",  "\n", TRUE);
  Text claLst = TxtBetween2Tag(filHea, "CLASS: ",   "\n", TRUE);
  Text verLst = TxtBetween2Tag(filHea, "VERSION: ", "\n", TRUE);
  Text purTxt = TxtBetween2Tag(filHea, "PURPOSE: ", DctTolSepBas, FALSE);
  Text purCls = ReplaceTable(purTxt, [[ [["\n// ",  "\n"]],
                                        [["_\n",    DctBr2+"\n"]] ]]);
  Text remTxt = TxtBetween2Tag(filHea, "COMMENT: ", DctTolSepBas, FALSE);
  Text remCls = Replace(Replace(remTxt, "\n// ", "\n"), "\n//", "\n");

  [[ filNam, autIde, claLst, verLst, purCls, remCls ]]
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto con los 4 atributos de la cabecera de un fichero que
son el nombre del fichero, el autor, sus clases y su proposito.",
DctGetHeader);
////////////////////////////////////////////////////////////////////////
"
```

Set DctGetHeaderRobust()

```
////////////////////////////////////////////////////////////////////////
Set DctGetHeaderRobust(Text codHea, // Trozo de codigo para extraer atributos
                       Text filNam, // Nombre del fichero si no se encuentra
                       Text filPur) // Descripcion si no se encuentra
////////////////////////////////////////////////////////////////////////
{
Text WriteLn(filPur);
  Real posFil = TextFind(codHea, " FILE",    0);      // Nombre del fichero
  Real posDts = TextFind(codHea, ": ",       posFil); // : tras el nombre
  Real posPur = TextFind(codHea, " PURPOSE", posDts); // Proposito tras :
```

```
  Set  filAtr = If(And(posFil, posDts, posPur, LE(posPur, 500)),
                DctGetHeader(codHea),   // Parece una cabecera
                [["","","","","","",""]]); // No parece una cabecera

  Text filAut = "http://www.asolver.com";
  Text chkNam = If(filAtr[1]=="", filNam, filAtr[1]);
  Text chkAut = If(filAtr[2]=="", filAut, filAtr[2]);
  Text chkCla = filAtr[3];
  Text chkVer = filAtr[4];
  Text chkPur = If(filAtr[5]=="", filPur, filAtr[5]);
  Text chkRem = filAtr[6];

  [[ chkNam, chkAut, chkCla, chkVer, chkPur, chkRem ]]
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto con los 4 atributos de la cabecera de un fichero que
son el nombre del fichero, el autor, sus clases y su proposito.",
DctGetHeaderRobust);
////////////////////////////////////////////////////////////////////////////
```

Set DctTolSplit()

```
////////////////////////////////////////////////////////////////////////////
Set DctTolSplit(Text codInp) // Codigo Tol
////////////////////////////////////////////////////////////////////////////
{
  Set  setTop = TxtTokenizer(codInp, DctTolSepTop); // Rotura de alto nivel

  Set  treTop = DctGetHeader(setTop[1]); // Cabecera del fichero

  Set  treSec = For(2, Card(setTop), Set(Real posSec)
  {
    Text codSec = setTop[posSec]+"\n"; // Ultima linea con su salto de linea
    Real lenSec = TextLength(codSec);
    Real heaEnd = TextFind(codSec, DctTolSepBas);
    Text heaSec = Compact(Sub(codSec, 1, heaEnd-1));
    Text decSec = Sub(codSec, heaEnd+TextLength(DctTolSepBas), lenSec);
    Set  decSet = TxtTokenizer("\n\n"+decSec, DctTolSepLow);

    If(EQ(Card(decSet),1), [[heaSec, SHiCompact(SHiTolPre(decSec))]], // Bruto
    { // Codigo correctamente delimitado
      Set  htmSet = For(2, Card(decSet), Set(Real posCod)
      {
        Text decCod = DctTolSepLow+decSet[posCod];
        Set  decInf = DctGetInterface(decCod); // 5 elementos

        decInf << SetOfText(SHiCompact(SHiTolPre(decCod))) // 6 elementos
      });
      [[heaSec, htmSet]]
    })
  });

  Set  treAll = [[ treTop ]] << treSec;

  treAll
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto de secciones (FILE, CONSTANTS, FUNCTIONS, etc.) resultado
de romper el codigo Tol que recibe como parametro por secciones.
Cada seccion esta formada por un conjunto con su nombre y su conjunto de
declaraciones, salvo:
- la primera que contiene la información general de la cabecera y
- las secciones de codigo en bruto que retorna un conjunto formado por el
  texto nombre de la foncion y un texto (no un conjunto) que es el codigo
  en bruto formateado.",
DctTolSplit);
////////////////////////////////////////////////////////////////////////////
```

Real DctRawCode()

```
////////////////////////////////////////////////////////////////////////////
Real DctRawCode(Set dctSec) // Seccion de declaracion
////////////////////////////////////////////////////////////////////////////
{ If(EQ(Card(dctSec), 2), Grammar(dctSec[2])!="Set", TRUE) };
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna cierto si el segundo componente no existe o si existe no es un
conjunto de declaraciones de codigo. Lo natural en caso de codigo bruto es
que exista pero sea un texto con dicho codigo en bruto.",
DctRawCode);
////////////////////////////////////////////////////////////////////////////
```

Text DctLink()

```
////////////////////////////////////////////////////////////////////////////
Text DctLink(Text prgNam, // Nombre del programa
             Text filNam, // Nombre del fichero
             Text pstSec, // Seccion
             Text ctrRet) // Si F->fichero, L->link, A->a href y T A+title
////////////////////////////////////////////////////////////////////////////
{
  Text filChk = If(filNam=="", "index", filNam); // No deberia ser vacio
  Text filExt = GetFileExtension(filChk);        // Extension
  Text filUrl =Case(
    filExt<:[["pdf","html"]],
      FilDelExtLow(prgNam)+"/"+filChk, // Pdf, Html no se cambian
    filExt=="htm",
      FilDelExtLow(prgNam)+"/"+filChk+"l",  // Poner la l de html
    TRUE,
      FilDelExtLow(prgNam +"/"+filChk)+".html");   // En general

  If(ctrRet=="F", "web/"+filUrl,
  {
    Text secUrl = TxtCat("../"+filUrl, "#", Htm2Ide(prgNam, pstSec, FALSE));
    Text secTit = pstSec+" en "+prgNam;
    Case(
      ctrRet=="L", secUrl,
      ctrRet=="A", "<a href="+Q(secUrl)+">",
      ctrRet=="T", "<a href="+Q(secUrl)+" title="+Q(secTit)+">",
      TRUE,        secUrl)
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna:
- si ctrRet=F el fichero de salida Html, salvo los Pdf que son Pdf,
- si ctrRet=L el enlace global al programa, fichero y seccion y
- si ctrRet=A el tag a href al enlace global.
Si no hay fichero filNam asume la pagina principal index, lo que no tendria
que suceder.
Si no hay seccion no añade el #seccion.",
DctLink);
////////////////////////////////////////////////////////////////////////////
```

Text DctPstReplace()

```
////////////////////////////////////////////////////////////////////////////
Text DctPstReplace(Text prgNam, // Nombre del programa
                   Text filNam, // Nombre del fichero
                   Text pstSec, // Seccion, para local, global y header
                   Text addCla, // Clase adicionales
                   Text pstSta, // Estado
                   Text pstAut, // Autor
                   Text pstDes, // Descripcion
                   Text pstHtm) // Html del post
```

```
////////////////////////////////////////////////////////////////////////
{
  Text pstPth = DctLink(prgNam, filNam, pstSec, "F"); // Fichero de salida
  Text pstLnk = DctLink(prgNam, filNam, pstSec, "L"); // Link a url

  Text pstCat = If(pstSec=="Presentación", "Escrito", "Código");
  Text topCla = pstCat+"; " + prgNam + "; " +  // Todas existen
                If(filNam==pstSec, filNam, filNam+"; "+pstSec) + "; " +
                DctFileLanguage(filNam,FALSE); // Lenguaje de programacion
  Text pstCla = TxtCat(topCla, "; ", addCla);

  Text lblGlo = pstSec+" de "+prgNam;
  Text wraDes = TxtLineWrap(pstDes, DctDesLen, TRUE)[1];
  Text nowFmt = DctTdy+" ["+FormatReal(Time, "%9.4lf")+"]"; // [9999.9999]

  // Eliminar los propios autoseparadores que puede contener pstHtm
  Text clsHtm = Replace(pstHtm, DctUndLinAsc, DctUndLinHtm);

  ReplaceTable(DctPstSee, [[ // Siempre se usa la misma semilla
    [["_CLA_",  pstCla]],
    [["B",  pstSta]],
    [["_DTE_",  nowFmt]], // Fecha DctTdy
    [["_AUT_",  pstAut]],
    [["web/dct_writer/dct_tol.html",  pstPth]], // Ruta del fichero Html de sa
    [["../dct_writer/dct_tol.html#Dct_Writer_Text_DctPstReplace__",  pstLnk]],
    [["_GLO_",  lblGlo]], // Titulo ventana, enlaces y headers globales
    [["Text DctPstReplace()",  pstSec]], // Etiqueta a names, enlaces y header
    [["_DES_",  wraDes]], // Descripcion ajustada a tamaño
    [["_HTM_",  clsHtm]]  // Sin lineas de undersecore
  ]], 1) // 1 vuelta solo pues Htm puede contener etiquetas _XXX_ parecidas
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna por reemplazamiento un post con todos sus campos.",
DctPstReplace);
////////////////////////////////////////////////////////////////////////
```

Text DctSecReplace()

```
////////////////////////////////////////////////////////////////////////
Text DctSecReplace(Text pstSec) // Seccion
////////////////////////////////////////////////////////////////////////
{
  ReplaceTable(pstSec, [[
    [["INCLUDE",              "Inclusiones"]],
    [["COMMON",               "comunes"]],
    [["APPLICATION",          "de aplicación"]],
    [["BLACKBOARD",           "Pizarra"]],
    [["STRUCTS",              "Estructuras de datos"]],
    [["CONTROLS",             "Variables de control"]],
    [["CONSTANTS",            "Constantes"]],
    [["SHORTNAMES",           "Funciones de nombre corto"]],
    [["FUNCTIONS",            "Funciones"]],
    [["MAKE",                 "Proceso"]],
    [["TEST",                 "Pruebas"]],
    [["END",                  "Finalización"]]
  ]], 1)
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna los nombres en castellano de los diferentes tipos de secciones.",
DctSecReplace);
////////////////////////////////////////////////////////////////////////
```

Text DctFileLanguage()

```
////////////////////////////////////////////////////////////////////////
Text DctFileLanguage(Text filNam, // Nombre del fichero de entrada
                     Real ctrLar) // Si true largo, si false corto=extension
```

```
////////////////////////////////////////////////////////////////////////
{
  Text filExt = ToLower(GetFileExtension(filPth));
  If(ctrLar,
    Case(
      filExt == "tol", "Time oriented language",
      filExt == "bst", "Datos con estructura de Tol",
      filExt == "bdt", "Datos de series temporales",

      filExt == "bat", "Command bat",
      filExt == "css", "Cascade style sheet",
      filExt == "gpl", "Gnuplot",
      filExt == "js",  "Javascript",
      filExt == "sql", "Standard query language",
      filExt == "xml", "Extensible markup language",
      filExt == "htm", "HyperText markup language",
      filExt == "html","HyperText markup language",
      filExt == "age", "Agenda de posts",

      filExt == "gif",  "Graphic file Gif format",
      filExt == "jpg",  "Graphic file Jpe format",
      filExt == "png",  "Graphic file Png format",

      TRUE,             "Ascii code"),
    Case(
      filExt == "bat", "Cmd",
      filExt == "htm", "Html",
      TRUE,            FirstToUpper(filExt)))
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el nombre del lenguaje de programación en formato corto o largo
dependiendo de ctrLar.",
DctFileLanguage);
////////////////////////////////////////////////////////////////////////
```

Set DctDeclare()

```
////////////////////////////////////////////////////////////////////////
Set DctDeclare(Text prgNam, // Programa
               Text filNam, // Fichero
               Text decCat, // Categoria: variable, funcion
               Text decTyp, // Tipo
               Text decNam, // Nombre
               Text decArg, // Argumentos si procede
               Text ascDes) // Descripcion en ascii, puede contener <, > o &
////////////////////////////////////////////////////////////////////////
{
  Text opeClo = If(decCat=="variable", "", "()");
  Text uppCat = FirstToUpper(decCat);
  Text simTol = decTyp+" "+decNam+opeClo; // Simula una declaracion Tol
  Text decLnk = DctLink(prgNam, filNam, simTol, "A"); // A href y url
  Text decTit = DctLink(prgNam, filNam, simTol, "T"); // A href url y title

  Text decDes = Compact(HtmAsc2Xml(ascDes));

  SetOfText(
    simTol,                                    // Tipo var o fun()
    uppCat+"; "+decTyp+"; "+decNam,            // Clases
    decTyp+" "+decNam+decArg+" "+decDes,       // Descripciones
    "  <li><code>"+decTyp+" "+decLnk+          // Para indices
                   decNam+"</a>"+
                   decArg+"</code><br />"+
                   decDes+"</li>\n",
    decTit+decNam+"</a>")                      // Para inyectar en otro codigo
};
////////////////////////////////////////////////////////////////////////
```

```
PutDescription(
"Para una declaracion Tol retorna un set de 5 elementos formado por:
a) clases: Variable/Funcion, tipo Tol y nombre
b) declaracion: tipo nombre(argumentos)
c) descripcion: tipo nombre(argumentos) y proposito
d) codigo para un li con tipo nombre(argumentos) y proposito y
e) a href link para que el codigo de otras funciones le enlacen.",
DctDeclare);
////////////////////////////////////////////////////////////////////////////
```

Real DctAppendPost()

```
////////////////////////////////////////////////////////////////////////////
Real DctAppendPost(Text outFil, // Semilla
                   Text prgNam, // Nombre del programa
                   Text filNam, // Nombre del fichero
                   Text pstSec, // Seccion, para local, global y header
                   Text addCla, // Clase adicionales
                   Text pstSta, // Estado
                   Text pstAut, // Autor
                   Text pstDes, // Descripcion
                   Text pstHtm) // Html del post
////////////////////////////////////////////////////////////////////////////
{
  Text putSep = If(TextBeginWith(addCla, "Cabecera"), "", DctPstSep);

  Text pstTxt = DctPstReplace(prgNam, filNam, pstSec, addCla, pstSta,
                              pstAut, pstDes, pstHtm); // Htm

  Text AppendFile(outFil, putSep+pstTxt);
  TRUE
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Añade un post con todos sus campos al fichero de agenda llamado outFil.
La cabecera no lleva separador, el resto es separador mas contenido.",
DctAppendPost);
////////////////////////////////////////////////////////////////////////////
```

Real DctWriteHeader()

```
////////////////////////////////////////////////////////////////////////////
Real DctWriteHeader(Text prgNam, // Nombre del progama
                    Text filNam, // Fichero
                    Set  dctCab, // Atributos de la cabecera del fichero
                    Text agePth, // Agenda
                    Text levCla) // Clase inicial de nivel
////////////////////////////////////////////////////////////////////////////
{
  Text WriteFile(agePth, ""); // El header inicializa el fichero

  Text cabCla = "Cabecera; " + TxtCat(levCla, "; ", dctCab[3]);

  Text cabIni = HtmAsc2Xml(dctCab[5]);
  Text cabDes = Compact(Replace(cabIni, DctBr2, " "));
  Text cabHtm = "<p>\n"+Replace(cabIni, DctBr2, "<br /><br />")+"\n</p>\n";

  DctAppendPost(agePth,     // Fichero de salida
                prgNam,     // Nombre del programa
                filNam,     // Nombre del fichero
                filNam,     // Seccion, para local, global y header
                cabCla,     // Mas las clases de la cabecera del fichero
                "D",        // Estado
                dctCab[2],  // Autor
                cabDes,     // Descripcion
                cabHtm)     // Html del post
};
////////////////////////////////////////////////////////////////////////////
```

```
PutDescription(
"Inicia la agenda escribiendo el post de la cabecera de un programa.",
DctWriteHeader);
////////////////////////////////////////////////////////////////////////
```

Real DctWriteTree()

```
////////////////////////////////////////////////////////////////////////
Real DctWriteTree(Text prgNam, // Nombre del progama
                  Text filNam, // Fichero
                  Set  dctCab, // Atributos de la cabecera del fichero
                  Text agePth, // Agenda
                  Set  inpTre, // Arbol de entrada
                  Text levCla) // Clase inicial de nivel
////////////////////////////////////////////////////////////////////////
{
  Text treCla = TxtCat("Árbol", "; ", TxtCat(levCla, "; ", dctCab[3]));
  Text pstHtm = DTrHtml(inpTre, prgNam, 0, filNam); // Arbol de ficheros
  Text pstDes = "Árbol de directorios y ficheros del programa "+prgNam+", "+
                "al que pertenece el fichero de código fuente "+filNam+". "+
                dctCab[5];

  DctAppendPost(agePth,              // Fichero de salida
                prgNam,              // Nombre del programa
                filNam,              // Nombre del fichero
                "Árbol de ficheros", // Seccion, para local, global y header
                treCla,              // Mas las clases de la cabecera
                "C",                 // Estado
                dctCab[2],           // Autor
                pstDes,              // Descripcion
                pstHtm)              // Html del post
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Escribe el post del arbol de navegacion de un programa con el fichero en
curso activo.",
DctWriteTree);
////////////////////////////////////////////////////////////////////////
```

Real DctWriteDisplay()

```
////////////////////////////////////////////////////////////////////////
Real DctWriteDisplay(Text prgNam, // Nombre del progama
                     Text filNam, // Fichero
                     Set  dctCab, // Atributos de la cabecera del fichero
                     Text agePth) // Agenda
////////////////////////////////////////////////////////////////////////
{
  Text htmPth = Replace(agePth, ".age", ".htm");
  Text iniTag = "<body><div class="+Q("Bdy")+">";
  Text htmCod = FilBetween2Tag(htmPth, iniTag, "</div></body>");
  Text pstDes = "Presentación del fichero de código fuente "+filNam+" "+
                "perteneciente al programa "+prgNam+". "+
                dctCab[5];

  If(htmCod=="", FALSE,
     DctAppendPost(agePth,          // Fichero de salida
                   prgNam,          // Nombre del programa
                   filNam,          // Nombre del fichero
                   "Presentación", // Seccion, para local, global y header
                   dctCab[3],       // Mas las clases de la cabecera
                   "C",             // Estado
                   dctCab[2],       // Autor
                   pstDes,          // Descripcion
                   htmCod))         // Html del post
};
////////////////////////////////////////////////////////////////////////
```

```
PutDescription(
"Escribe el post de la presentación realizada a mano de un fichero filNam de
un programa prgNam.",
DctWriteDisplay);
////////////////////////////////////////////////////////////////////////////
```

Real DctWriteIndex()

```
////////////////////////////////////////////////////////////////////////////
Real DctWriteIndex(Text prgNam, // Nombre del progama
                   Text filNam, // Fichero
                   Set  dctSHi, // Codigo cortado y realzado
                   Text agePth, // Agenda
                   Text funDmp) // Fichero de volcado de funciones
////////////////////////////////////////////////////////////////////////////
{
  Set  dctCab = dctSHi[1]; // Atributos de cabecera del fichero
  Text pstDes = "Índice de las variables y funciones declaradas en "+
                "el fichero de código fuente "+filNam+" "+
                "del programa "+prgNam+". "+dctCab[5];

  // Indice de variables y funciones
  Set  dctInd = For(2, Card(dctSHi), Text(Real dctPos)
  {
    Set  dctSec = dctSHi[dctPos];
    If(dctSec[1]=="END", "", // A partir del end no ha de haber nada
    {
      Text secCls = DctSecReplace(dctSec[1]); // Nombre de la seccion

      If(DctRawCode(dctSec), "", // No tiene declaraciones o son en bruto
      {
        Set  decLst = dctSec[2]; // Lista de declaraciones de la seccion
        Set  decCic = EvalSet(decLst, Text(Set tolSet) // Por declaracion
        {
          Text If(dctSec[1]!="FUNCTIONS", "",          // No es funcion larga
                  AppendFile(funDmp, tolSet[3]+"\n")); // Dump de la funcion

          Set  decSet = DctDeclare(prgNam, filNam, // Proyecto y fichero
                                   tolSet[1],      // Variable, funcion
                                   tolSet[2],      // Tipo, gramatica
                                   tolSet[3],      // Nombre
                                   tolSet[4],      // Argumentos si procede
                                   tolSet[5]);     // Descripcion de funcion
          decSet[4] // Es el li con enlace y resumen
        });
        "<p><b>"+secCls+"</b></p>\n<ul>\n"+SetSum(decCic)+"</ul>\n"
      })
    })
  });
  Text htmInd = SetSum(dctInd); // Une los textos de todas las secciones

  DctAppendPost(agePth,          // Fichero de salida
                prgNam,          // Nombre del programa
                filNam,          // Nombre del fichero
                "Declaraciones", // Seccion, para local, global y header
                dctCab[3],       // Mas las clases de la cabecera
                "C",             // Estado
                dctCab[2],       // Autor
                pstDes,          // Descripcion
                htmInd)          // Html del post
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Escribe el post del indice de declaraciones de variables y funciones de un
fichero filNam de un programa prgNam.
Es un indice con enlaces al resto de declaraciones, pero se le titula como
declaraciones pues no indexa otras secciones.",
DctWriteIndex);
////////////////////////////////////////////////////////////////////////////
```

Real DctWriteCode()

```
////////////////////////////////////////////////////////////////////////////
Real DctWriteCode(Text prgNam, // Nombre del progama
                  Text filNam, // Fichero
                  Set  dctSHi, // Codigo cortado y realzado
                  Text agePth, // Agenda
                  Set  lnkTab) // Tabla de declaraciones y sus enlaces
////////////////////////////////////////////////////////////////////////////
{
  Set  dctCab = dctSHi[1]; // Atributos de cabecera del fichero

  // Escribir todas las secciones con todas sus declaraciones
  Set  dctCic = For(2, Card(dctSHi), Real(Real dctPos)
  {
    Set  dctSec = dctSHi[dctPos];
    Text secCls = DctSecReplace(dctSec[1]);         // Seccion en castellano
    Text secCla = FirstToUpper(ToLower(dctSec[1])); // Seccion en ingles
    Text pstDes = secCls+" ("+dctSec[1]+") del fichero de código fuente "+
                  filNam+" del programa "+prgNam+". "+
                  dctCab[5];

    Real notFmt = DctRawCode(dctSec); // Codigo bruto, sin formato
    Text secCod = If(!notFmt, "",     // El codigo se pone luego
                     HtmPutLinks(dctSec[2], lnkTab, "")); // Bruto+links

    DctAppendPost(agePth,     // Fichero de salida
                  prgNam,     // Nombre del programa
                  filNam,     // Nombre del fichero
                  secCls,     // Seccion, para local, global y header
                  secCla,     // Clase = seccion en formato original
                  "C",        // Estado
                  dctCab[2],  // Autor
                  pstDes,     // Descripcion
                  secCod);    // Solo si es codigo bruto

    If(notFmt, FALSE,         // Sin formato->sin ciclo
    {
      Set  decLst = dctSec[2]; // Lista de declaraciones de la seccion

      Set  decCic = EvalSet(decLst, Real(Set tolSet) // Para cada declaracion
      {
        Set decSet = DctDeclare(prgNam, filNam, // Proyecto y fichero
                                tolSet[1],      // Categoria: variable, funcio
                                tolSet[2],      // Tipo, gramatica
                                tolSet[3],      // Nombre
                                tolSet[4],      // Argumentos si procede
                                tolSet[5]);     // Descripcion de la funcion

        Text codImg = HtmPutCharts(tolSet[6]);  // Añade graficos online
        Text codLnk = HtmPutLinks (codImg, lnkTab, "");   // Poner links

        DctAppendPost(agePth,     // Fichero de salida
                      prgNam,     // Nombre del programa
                      filNam,     // Nombre del fichero
                      decSet[1],  // Type var o type fun()
                      decSet[2],  // Clases + nombre
                      "B",        // Estado
                      dctCab[2],  // Autor
                      dctCab[3],  // Descripcion tipo var/fun arg purpose
                      codLnk)     // Codigo realzado y con links
      });
      Card(decCic)
    })
  });
  Card(dctCic)
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Escribe todos los posts de secciones de codigo (constantes, funciones, etc.)
y todos los post de cada una de las estructuras, constantes, funciones, etc.
de un fichero filNam de un programa prgNam.",
DctWriteCode);
```

```
////////////////////////////////////////////////////////////////////////
```

Real DctWriteSource()

```
////////////////////////////////////////////////////////////////////////
Real DctWriteSource(Text prgNam, // Nombre del progama
                    Text filNam, // Fichero
                    Set  dctCab, // Atributos de la cabecera del fichero
                    Text agePth, // Agenda
                    Text codTxt, // Codigo fuente del fichero, total o parcial
                    Set  lnkTab, // Tabla de declaraciones y sus enlaces
                    Text codDmp) // Fichero de volcado de codigo
////////////////////////////////////////////////////////////////////////
{
  Text AppendFile(codDmp,"\n"+DctUndLinAsc+"\n"+filNam+"\n"+codTxt); // Dump

  If(Compact(codTxt)=="", FALSE, // No hay codigo que escribir
  {
    Text lanLar = DctFileLanguage(filNam, TRUE);  // Nombre largo del lenguaje
    Text lanTin = DctFileLanguage(filNam, FALSE); // Nombre corto del lenguaje
    Text lanCla = TxtCat("Fuente",  "; ", lanTin);
    Text allCla = TxtCat(dctCab[3], "; ", lanCla);

    Text cabNam = dctCab[1]; // Nombre del fichero segun la cabecera
    Text lanTst = If(filNam==cabNam, lanLar, "¿"+filNam+" o "+cabNam+"?");

    Text pstDes = "Código fuente en lenguaje "+lanLar+" ("+lanTin+") "+
                  "del fichero "+filNam+" del programa "+prgNam+". ";

    Text notMai = Replace(codTxt, " asalmeronasolver.com", ""); // Eliminable

    Text codSHi = SHiPreCode  (filNam, notMai);     // Realza el texto
    Text codImg = HtmPutCharts(codSHi);             // Añade graficos online
    Text codLnk = HtmPutLinks (codImg, lnkTab, ""); // Poner links

    DctAppendPost(agePth,           // Fichero de salida
                  prgNam,           // Nombre del programa
                  filNam,           // Nombre del fichero
                  lanTst,           // Seccion, para local, global y header
                  allCla,           // Mas las clases de la cabecera
                  "C",              // Estado
                  dctCab[2],        // Autor
                  pstDes,           // Descripcion
                  codLnk)           // Html del post codigo realzado con links
  })
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Escribe el post de la presentación realizada a mano de un fichero filNam de
un programa prgNam.",
DctWriteSource);
////////////////////////////////////////////////////////////////////////
```

Real DctMake()

```
////////////////////////////////////////////////////////////////////////
Real DctMake(Text dctPre, // Directorio contenedor del directorio del programa
             Set  dctTre) // Arbol de ficheros a documentar
////////////////////////////////////////////////////////////////////////
{
  Text WriteLn(DTrText(dctTre, "", 0)); // Test visual previo

  Text prgNam = Anything A(dctTre[1],                    "Program.........");
  Text prgPth = Anything A(TxtCat(dctPre, "/", prgNam), "Input directory.");
  Text outPth = Anything A(TxtCat(prgPth, "/", "dct"),  "Output directory");

  // Ficheros de volcado completo
  Text funDmp = Anything A(outPth+"/_dmpfun.txt",       "Func dump file..");
  Text codDmp = Anything A(outPth+"/_dmpcod.txt",       "Code dump file..");
```

```
  Text WriteFile(funDmp, "");
  Text WriteFile(codDmp, "");

  Set  filTab = DtrFileExc(DtrFileSet(dctTre, dctPre)); // Tabla de ficheros

  Set  lnkTab = DctLinkAllDeclaration(filTab); // Tabla (declaracion, enlace)

  Set  filCic = EvalSet(filTab, Real(Set filSet)
  {
    Text filNam = Anything_A(filSet[3], "File to agenda..");
    Text filPth = filSet[2];
    Text filPur = FirstToUpper(filSet[4]); // Descripcion por defecto

    Text agePth = outPth+"/"+FilDelExtLow(filNam)+".age";

    Text tagIni = filSet[5]; // Tag inicial si lo hubiera
    Text tagEnd = filSet[6]; // Tag inicial si lo hubiera

    Text filCod = FilBetween2Tag(filPth,  tagIni, tagEnd);

    If(TextEndAt(filNam, ".tol"), // Ficheros Tol
    {
      Text levCla = If(filSet[1]=="Rot", "Raíz", "Hoja");
      Set  dctSHi = DctTolSplit(filCod); // Corta y realza el codigo
      Set  dctCab = dctSHi[1];           // Atributos de cabecera del fichero

      DctWriteHeader (prgNam, filNam, dctCab, agePth, levCla);

      If(filSet[1]!="Rot", FALSE, // Si no es raiz el arbol al final
         DctWriteTree   (prgNam, filNam, dctCab, agePth, dctTre, levCla));

      DctWriteDisplay(prgNam, filNam, dctCab, agePth);
      DctWriteIndex  (prgNam, filNam, dctSHi, agePth, funDmp);
      DctWriteCode   (prgNam, filNam, dctSHi, agePth, lnkTab);
      DctWriteSource (prgNam, filNam, dctCab, agePth, filCod, lnkTab, codDmp);

      If(filSet[1]=="Rot", FALSE,
         DctWriteTree   (prgNam, filNam, dctCab, agePth, dctTre, levCla))
    },
    If(TextEndAt(filNam, ".pdf"), FALSE, // Archivos Pdf, nada que hacer
    {
      // Puede haber codigo Tol en ficheros htm, no html
      Set  lnkChk = If(TextEndAt(filNam, ".htm"), lnkTab, Empty);
      // Los ficheros no Tol pueden no tener buenas cabeceras
      Set  dctCab = DctGetHeaderRobust(filCod, filNam, filPur); // Atributos

      DctWriteHeader (prgNam, filNam, dctCab, agePth, "Hoja");
      DctWriteDisplay(prgNam, filNam, dctCab, agePth);
      DctWriteSource (prgNam, filNam, dctCab, agePth, filCod, lnkChk, codDmp);
      DctWriteTree   (prgNam, filNam, dctCab, agePth, dctTre, "Hoja")
    }))
  });

  Set DctOcurrences(funDmp, codDmp, 2);

  Card(filCic)
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Proceso principal de documentacion de todo el arbol de fichero de entrada.",
DctMake);
////////////////////////////////////////////////////////////////////////////
```

Set DctOcurrences()

```
////////////////////////////////////////////////////////////////////////////
Set DctOcurrences(Text funDmp, // Fichero de volcado de funciones
                  Text codDmp, // Fichero de volcado de codigo
                  Real ctrOcc) // Control de ocurrencias
////////////////////////////////////////////////////////////////////////////
{
  Set  funSet = Select(Tokenizer(ReadFile(funDmp),"\n"), Real(Text funNam)
```

```
                { Compact(funNam)!="" });
  Text codAll = ReadFile(codDmp);
  Set  funTab = EvalSet(funSet, Set(Text funNam)
                { [[funNam, TextOccurrences(codAll, funNam)]] });
  Set  funFew = Select(funTab, Real(Set funOcc) { LT(funOcc[2], ctrOcc) });
  If(!Card(funFew), Empty,
  {
    Text WriteLn("\nOcurrences:");
    EvalSet(funFew, Set(Set funOcc)
    {
      Text WriteLn("->"+funOcc[1]+": "+F(funOcc[2]));
      funOcc
    })
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna una tabla de funciones y numero de occurencias para aquellas
funciones que no superan ctrOcc, como efecto latelar las visualiza.",
DctOcurrences);
////////////////////////////////////////////////////////////////////////////
```

Set DctLinkFileDeclaration()

```
////////////////////////////////////////////////////////////////////////////
Set DctLinkFileDeclaration(Text prgNam, // Nombre del progama
                           Text filNam, // Fichero
                           Set  dctSHi) // Codigo cortado y realzado
////////////////////////////////////////////////////////////////////////////
{
  // Para todas las variables y funciones
  Set  lblCic = For(2, Card(dctSHi), Set(Real dctPos)
  {
    Set  dctSec = dctSHi[dctPos];
    If(DctRawCode(dctSec), Empty, // No tiene declaraciones o son en bruto
    {
      Set  decLst = dctSec[2]; // Lista de declaraciones de la seccion
      Set  decCic = EvalSet(decLst, Set(Set tolSet) // Por declaracion
      {
        Set  decSet = DctDeclare(prgNam, filNam, // Proyecto y fichero
                                 tolSet[1],      // Variable, funcion
                                 tolSet[2],      // Tipo, gramatica
                                 tolSet[3],      // Nombre
                                 tolSet[4],      // Argumentos si procede
                                 tolSet[5]);     // Descripcion de funcion
        Text decNam = tolSet[3]; // Nombre de la funcion o variable
        Text decLnk = decSet[5]; // a href a la funcion o variable
        SetOfText(decNam,decLnk) // Par (declaracion, enlace)
      })
    })
  });
  BinGroup("<<", lblCic)
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna una tabla de pares nombre de variable o funcion y su enlace de un
fichero Tol de un programa.",
DctLinkFileDeclaration);
////////////////////////////////////////////////////////////////////////////
```

Set DctLinkAllDeclaration()

```
////////////////////////////////////////////////////////////////////////////
Set DctLinkAllDeclaration(Set  filTab) // Tabla de ficheros, selecciona Tol
////////////////////////////////////////////////////////////////////////////
{
  Set  lnkCic = EvalSet(filTab, Set(Set filSet)
  {
    Text filNam = Anything A(filSet[3], "File get links..");
```

```
    If(!TextEndAt(filNam, ".tol"), Empty, // No es fichero Tol
    {
      Text tagIni = filSet[5]; // Tag inicial si lo hubiera
      Text tagEnd = filSet[6]; // Tag inicial si lo hubiera
      Text filPth = filSet[2]; // Ruta del fichero
      Text filCod = FilBetween2Tag(filPth,  tagIni, tagEnd); // Lee codigo
      Set  dctSHi = DctTolSplit(filCod); // Corta y realza el codigo
      DctLinkFileDeclaration(prgNam, filNam, dctSHi) // Tabla (declara,enlace)
    })
  });
  BinGroup("<<", lnkCic)
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna una tabla de pares nombre de variable o funcion y su enlace de todos
los ficheros Tol de un programa.",
DctLinkAllDeclaration);
////////////////////////////////////////////////////////////////////////////
```

inc.tol de Dct.Writer

Inclusion de ficheros de funciones comunes y basicas y de funciones especificas de aplicacion

Declaraciones

Inclusiones comunes

- `Set txtInc`
  Incluir funciones de textos.
- `Set anyInc`
  Incluir funciones de anything.
- `Set filInc`
  Incluir funciones de ficheros.
- `Set htmInc`
  Incluir funciones de Html.

Inclusiones de aplicación

- `Set shiInc`
  Incluir funciones de sintaxis realzada.
- `Set dtrInc`
  Incluir funciones de arboles para documentar.
- `Set dctInc`
  Incluir funciones de documentacion.

Set txtInc

```
////////////////////////////////////////////////////////////////////////////
Set  txtInc = Include("cmm/txt.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Incluir funciones de textos.", txtInc);
////////////////////////////////////////////////////////////////////////////
```

Set anyInc

```
////////////////////////////////////////////////////////////////////////////
Set  anyInc = Include("cmm/any.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Incluir funciones de anything.", anyInc);
////////////////////////////////////////////////////////////////////////////
```

Set filInc

```
////////////////////////////////////////////////////////////////////////////
Set  filInc = Include("cmm/fil.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Incluir funciones de ficheros.", filInc);
////////////////////////////////////////////////////////////////////////////
```

Set htmInc

```
////////////////////////////////////////////////////////////////////////////
Set  htmInc = Include("cmm/htm.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Incluir funciones de Html.", htmInc);
////////////////////////////////////////////////////////////////////////////
```

Set shiInc

```
////////////////////////////////////////////////////////////////////////////
Set  shiInc = Include("app/shi.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Incluir funciones de sintaxis realzada.", shiInc);
////////////////////////////////////////////////////////////////////////////
```

Set dtrInc

```
////////////////////////////////////////////////////////////////////////////
Set  dtrInc = Include("app/dtr.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Incluir funciones de arboles para documentar.", dtrInc);
////////////////////////////////////////////////////////////////////////////
```

Set dctInc

```
////////////////////////////////////////////////////////////////////////////
Set  dctInc = Include("app/dct.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Incluir funciones de documentacion.", dctInc);
////////////////////////////////////////////////////////////////////////////
```