FIE BDC

Formato de Intercambio Estandar | Bases de Datos de Construcción

PROPUESTA DE

DEFINICION DEL FORMATO DE INTERCAMBIO ESTANDAR DE BASES DE DATOS DE CONSTRUCCION FIEBDC-3/98

INDICE:

PROPUESTA DE DEFINICION DEL FORMATO DE INTERCAMBIO ESTANDAR DE BASES DE DATOS DE CONSTRUCCION FIEBDC-3/98

PRESENTACION

La siguiente DEFINICION del Formato de Intercambio ESTANDAR de Bases de Datos de CONSTRUCCION, fue aprobada en abril de 1.977, para entrar en vigor a partir del 1 de enero de 1.998 como VERSION 3.

Este documento se pone a disposición de usuarios y empresas, con la única condición de que cualquier implementación informática del presente formato debe recoger tanto la entrada como la salida de datos.

Este formato pretende abarcar toda la INFORMACION contenida en las actuales bases de datos de CONSTRUCCION.

No todos los desarrolladores de bases de datos necesitarán utilizar todas las posibilidades del formato; así como tampoco todos los programas de mediciones y presupuestos harán uso de toda la INFORMACION suministrada.

Se prevé además, dentro del propio formato, la posibilidad de ampliación; manteniendo en lo posible la compatibilidad entre versiones en el caso de tratar nuevos contenidos que se prevean en un futuro.

FORMATO FIEBDC-3. ESPECIFICACION.

Toda la INFORMACION necesaria para reconstruir completamente una base de datos u obra en soportes físico y lógico distintos a aquellos en los cuales se produjo la INFORMACION es el objetivo del formato FIEBDC, Formato de Intercambio ESTANDAR de Bases de Datos de CONSTRUCCION.

La INFORMACION de una base de datos, obra o certificación se dispondrá en cualquier número de archivos en formato FIEBDC, con la extensión ".BC3", teniendo en cuenta que estos archivos ordenados alfabéticamente contengan la INFORMACION en el orden deseado.

La única limitación de tamaño de cada archivo será la máxima que permita el soporte físico utilizado para su transporte. Si se utiliza algún tipo de compresor de archivos, se deberá incluir en el mismo soporte el descompresor o utilizar un formato autodescomprimible.

El juego de caracteres a emplear en los campos CODIGO será el definido por MS-DOS 6.0, incluyendo < . > (ASCII-46), < $ > (ASCII-36), < # > (ASCII-35), < %> (ASCII-37), < & > (ASCII-38), < _ > (ASCII-95).

El fin de línea será el ESTANDAR de los archivos MS-DOS (ASCII-13 y ASCII-10). El fin de archivo se marcará según el mismo ESTANDAR (ASCII-26). El único carácter de control adicional que se permitirá será el tabulador (ASCII-9).

Cada archivo estará compuesto de registros, zonas de texto entre el carácter de principio de registro < ~ > (ASCII-126) y el siguiente principio de registro o fin de archivo. Los archivos deberán contener registros completos, es decir, la división de archivos se deberá realizar al comienzo de un registro (carácter < ~ >).

Cada registro estará compuesto de campos separados por caracteres < | > (ASCII-124). Todo campo con INFORMACION tendrá que finalizar con el separador de campos y el registro deberá contener todos los separadores de campos anteriores, aunque no contengan INFORMACION. No es necesario disponer de finalizadores de los campos posteriores al último con INFORMACION.

Cada campo estará compuesto de subcampos separados por caracteres < \ > (ASCII-92). El separador final, entre el último dato de un campo y el fin de campo es opcional.

El primer campo de cada registro es la cabecera de registro, una letra mayúscula que identifica el tipo de registro.

Se ignorará cualquier INFORMACION entre el último separador de campos de un registro (carácter < | >) o el comienzo del archivo y el comienzo del siguiente registro (carácter < ~>).

Se ignorarán los caracteres blancos (32), tabuladores (9) y de fin de línea (13 y 10), delante de los separadores < ~ >, < | > y < \ >.

No se podrán actualizar parcialmente campos de segundo orden (subcampos). Deberá actualizarse la INFORMACION completa de un campo en cualquiera de los registros

La disposición de registros dentro de un archivo es completamente libre, pero se garantizará la lectura secuencial de los mismos para evitar ambigüedades en las sustituciones de INFORMACION.

Los campos vacíos se considerarán SIN INFORMACION, no con INFORMACION nula, esto permite producir archivos de actualización que contengan únicamente la INFORMACION en alguno de sus campos y, por supuesto, el CODIGO de referencia.

Para anular un campo numérico deberá aparecer explícitamente el valor 0 (cero).

Para anular un campo alfanumérico deberá aparecer explícitamente el ROTULO NUL.

CONVENIOS DE NOTACION

[a] Indica nada o "a"

{a} Indica cero o más ocurrencias de "a"

(<DD>c) Tamaño máximo en número de caracteres del campo

Todos los valores numéricos irán sin separadores de miles y con el carácter punto ’.’ entre la parte entera y la decimal.

REGISTRO TIPO PROPIEDAD Y VERSION

Este registro se utiliza para documentar la procedencia y el formato de los archivos y, cuando exista, se dispondrá al comienzo del primer archivo.

~V | PROPIEDAD_ARCHIVO | VERSION_FORMATO \DDMMAA | PROGRAMA_EMISION | [CABECERA] \ { ROTULO_IDENTIFICACION \ } | JUEGO_CARACTERES |

PROPIEDAD_ARCHIVO: Redactor de la base de datos u obra, fecha, ...

VERSION_FORMATO: VERSION del formato del archivo, la actual es FIEBDC-3

DDMMAA: DD representa el día con dos dígitos, MM el mes y AA el año, si la fecha tiene menos de 5 dígitos representa mes y año únicamente, si tiene menos de tres, solo el año. Si se identifica la fecha con un número impar de dígitos, se completará con el carácter cero por la izquierda.

PROGRAMA_EMISION: Programa y/o empresa que genera los ficheros en formato BC3.

CABECERA: Título general de los ROTULOS_IDENTIFICACION.

ROTULO_IDENTIFICACION: Asigna secuencialmente títulos a los valores definidos en el campo PRECIO del registro ~C, que tal como se indica en su ESPECIFICACION, puede representar distintas épocas, ámbitos geográficos, etc., estableciéndose una relación biunívoca entre ambos.

JUEGO_CARACTERES: Asigna si el juego de caracteres a emplear es el definido para D.O.S., cuyos identificadores serán 850 ó 437, o es el definido para Windows, cuyo identificador será ANSI. En caso de que dicho campo esté vacío se interpretará, por omisión, que el juego de caracteres a utilizar será el 850 por compatibilidad con versiones anteriores.

REGISTRO TIPO CONCEPTO

Este registro contiene la INFORMACION básica de un concepto de cualquier tipo, material, auxiliar, partida, capítulo, entidad, documento, etc., tanto en su VERSION paramétrica como DEFINICION tradicional.

~C | {CODIGO \ } | UNIDAD | RESUMEN | {PRECIO \} | { FECHA \ } | TIPO |

CODIGO: CODIGO del concepto descrito. Un concepto puede tener varios CODIGOs que actuarán como sinónimos, este mecanismo permite integrar distintos sistemas de clasificación.

Para distinguir el concepto tipo raíz de un archivo, así como los conceptos tipo capítulo, se ampliará su CODIGO con los caracteres ’##’ y ’#’ respectivamente; quedando dicha NOTACION reflejada obligatoriamante en el registro tipo ~C ,siendo opcional en los restantes registros del mismo concepto.

Las referencias a un CODIGO con y sin # y ##, se entienden únicas a un mismo concepto.

Unicamente puede haber un concepto raíz en una base de datos u obra.

UNIDAD: Unidad de medida. Existe una relación de unidades de medida recomendadas, elaborada por la Asociación de Redactores de Bases de Datos de CONSTRUCCION.

RESUMEN: Resumen del texto descriptivo. Cada soporte indicará el número de caracteres que admite en su campo resumen.

PRECIO: Precio del concepto. Un concepto puede tener varios precios alternativos que representen distintas épocas, ámbitos geográficos, etc., definidos biunívocamente respecto al campo [CABECERA] \ { ROTULO_IDENTIFICACION \ } del registro ~V. Cuando haya más de un precio se asignarán secuencialmente a cada ROTULO definido; si hay más ROTULOS que precios, se asignará a aquellos el último precio definido.

FECHA: Fecha de la última actualización del precio. Cuando haya más de una fecha se asignarán secuencialmente a cada precio definido, si hay más precios que fechas, los precios sin su correspondiente fecha tomarán la última fecha definida.

Las fechas se definirán en el formato DDMMAA; DD representa el día con dos dígitos, MM el mes y AA el año, si la fecha tiene menos de 5 dígitos representa mes y año únicamente, si tiene menos de tres, solo el año. Si se identifica la fecha con un número impar de dígitos, se completará con el carácter cero por la izquierda.

TIPO: Tipo de concepto, Inicialmente se reservan los siguientes tipos:

0 (Sin clasificar) 1 (Mano de obra), 2 (Maquinaria y medios aux.), 3 (Materiales).

REGISTRO TIPO DESCOMPOSICION

Este registro contiene la descomposición de un concepto en otros a través de una o dos cantidades. El mismo registro lo emplearemos para definir la descomposición de un concepto tipo unidad de obra en conceptos tipo materiales, mano de obra, maquinaria y auxiliares y para la descomposición de un concepto tipo capítulo en conceptos tipo unidad de obra o subcapítulo.

~D | CODIGO_PADRE | { CODIGO_HIJO \ FACTOR \ RENDIMIENTO \ } |

CODIGO_PADRE: CODIGO del concepto descompuesto.

CODIGO_HIJO: CODIGO de cada concepto que interviene en la descomposición.

FACTOR: Factor de rendimiento, por defecto 1.0

RENDIMIENTO: Número de unidades, rendimiento o medición.

Cuando CODIGO_HIJO es un porcentaje, éste tiene tres partes:

1.- Prefijo que forma una máscara indicando sobre qué elementos se aplica el porcentaje. Si el prefijo es nulo, el porcentaje se aplica a todas las líneas anteriores.

2.- Un juego de caracteres que puede ser:

’&’ para porcentajes acumulables.

’%’ para porcentajes no acumulables expresados en tantos por uno.

3.- El resto del CODIGO permite diferenciar un porcentaje de otro.

Ejemplo: LD%N0001

LD - Sobre todas las líneas anteriores cuyo CODIGO comience por LD

% - No acumulable en tanto por uno.

N0001 - CODIGO diferenciador.

El rendimiento será el porcentaje que se aplica sobre las líneas anteriores a la actual y que queden afectadas por la máscara.

Ejemplo de una línea de descomposición: L%N004 \\0.03\

Esta línea representa un porcentaje del 0.03 por uno (3%) de todas las líneas anteriores a la actual, incluso porcentajes, cuyo CODIGO comience por L y cuyo texto estará en la DEFINICION del CODIGO ’L%N004’.

REGISTRO TIPO AÑADIR DESCOMPOSICION

Con este registro se pueden añadir líneas de descomposición, el registro tipo ~D cambia la descomposición completa. Para añadir conceptos nuevos a una base de datos, además de definir los registros C,T,L,D,... deberíamos posicionar los nuevos conceptos en el capítulo o capítulos donde queramos situarlos, para ello, es necesario un registro que nos permita añadir una o varias líneas de descomposición por cada capítulo donde queramos posicionar un nuevo concepto.

~Y | CODIGO_PADRE | { CODIGO_HIJO \ FACTOR \ RENDIMIENTO \ } |

Todos los campos tienen el mismo significado que en el registro tipo D.

REGISTRO TIPO TEXTO

Este registro contiene el texto descriptivo de un concepto

~T | CODIGO_CONCEPTO | TEXTO_DESCRIPTIVO |

CODIGO_CONCEPTO: CODIGO del concepto descrito

TEXTO_DESCRIPTIVO: Texto descriptivo del concepto sin limitación de tamaño. El texto podrá contener caracteres fin de línea (ASCII-13 + ASCII-10) que se mantendrán al reformatearlo.

REGISTRO TIPO DESCRIPCION PARAMETRICA

Este registro contiene la descripción paramétrica, bien en formato tradicional bien en formato API para DLL, que incluye la DEFINICION de parámetros, descomposiciones, comentario de ayuda a la selección de parámetros, resúmenes, textos, pliegos, claves e INFORMACION comercial, en función de tablas, expresiones y variables, para una familia de conceptos.

Este registro puede adoptar dos formas:

~P | CODIGO_FAMILIA | DESCRIPCION_PARAMETRICA |

Cuando CODIGO_FAMILIA está lleno, o bien DESCRIPCION_PARAMETRICA está llena, o bien DESCRIPCION_PARAMETRICA está vacía. En éste último caso se accede a la descripción paramétrica de la familia a través del archivo NOMBRE.DLL.

~P | | DESCRIPCION_PARAMETRICA | NOMBRE.DLL |

Cuando CODIGO_FAMILIA está vacío, se refiere al paramétrico global.

Si DESCRIPCION_PARAMETRICA está llena, el paramétrico global se establece a partir de ésta. Si DESCRIPCION_PARAMETRICA está vacía y NOMBRE.DLL está lleno, se establece a partir de éste. Si DESCRIPCION_PARAMETRICA y NOMBRE.DLL están llenos a la vez, tan solo es válida DESCRIPCION_PARAMETRICA.

CODIGO_FAMILIA: CODIGO del concepto tipo familia descrito. Si se utiliza un modelo de codificación dependiente de los parámetros (ver Anexos 2 y 3), este código debe poseer un carácter ?$? en su séptima posición, y los conceptos en los que se deriva tendrán como código los seis primeros caracteres del mismo más un carácter adicional por cada parámetro que posea.

DESCRIPCION_PARAMETRICA: Ver Anexo 2.

NOMBRE.DLL: Ver Anexo 3.

REGISTRO TIPO PLIEGOS

Este registro contiene las diferentes secciones y textos del pliego de condiciones de un concepto. El pliego de condiciones se estructura de forma jerárquica con el Sistema de Clasificación por Codificación y de forma facetada en varias secciones de distinto contenido.

SECCIONES DE LOS PLIEGOS.

Cuando el primer campo del registro ~L está vacío, el registro define los CODIGOs de las SECCIONES de cada pliego

sus ROTULOs correspondientes. Este registro es único para una base de datos u obra.

~L | | { CODIGO_SECCION_PLIEGO \ ROTULO_SECCION_PLIEGO \ } |

CODIGO_SECCION_ PLIEGO: CODIGO que define cada SECCION o faceta del pliego.

ROTULO_SECCION_PLIEGO: DEFINICION del ROTULO asociado a cada CODIGO correspondiente de cada SECCION o faceta del pliego.

Ejemplo de las secciones de los pliegos definidas para la Base de Datos de CONSTRUCCION de la Comunidad de Madrid y la Base de Datos de CONSTRUCCION de la Comunidad Valenciana, indicando CODIGO y ROTULO de la SECCION:

~L| | DES\ DESCRIPCION Y COMPLEMENTOS AL TEXTO

\PRE\ REQUISITOS PREVIOS A LA EJECUCIÓN

\COM\ COMPONENTES

\EJE\ EJECUCION Y ORGANIZACION

\NOR\ NORMATIVA

\CON\ CONTROL Y ACEPTACION

\SEG\ SEGURIDAD E HIGIENE

\VAL\ CRITERIOS DE VALORACION Y MEDICION

\MAN\ MANTENIMIENTO

\VAR\ VARIOS \ |

MODELO 1 DE TEXTOS DE LOS PLIEGOS.

Cuando el primer campo del registro ~L no está vacío, identifica a un concepto determinado. Puede haber un registro de este tipo por cada concepto de una base de datos u obra.

~L | CODIGO_CONCEPTO | {CODIGO_SECCION_PLIEGO \TEXTO_SECCION_PLIEGO \ } |

CODIGO_CONCEPTO: CODIGO del concepto descrito, contenido en la base de datos.

CODIGO_SECCION_PLIEGO: DEFINICION del CODIGO asociado a cada pliego.

TEXTO_SECCION_PLIEGO: Texto asignado a cada faceta o SECCION del pliego de condiciones del concepto.

El pliego de condiciones de cada concepto estará dividido con caracteres '\' en varias secciones o facetas, pensadas para imprimirse juntas o por separado.

Los fines de línea de cada SECCION del pliego se tratarán como en el REGISTRO TIPO TEXTO.

MODELO 2 DE TEXTOS DE LOS PLIEGOS.

Otra opción permite asignar el Pliego mediante párrafos de texto asociados a conceptos, utilizando el siguiente esquema de registros, como forma alternativa a la anterior:

~Q | { CODIGO_CONCEPTO \ } | { CODIGO_SECCION_PLIEGO \ CODIGO_PARRAFO

\ { ABREV_AMBITO; } \ } |

~J | CODIGO_PARRAFO | TEXTO_PARRAFO | TEXTO_PARRAFO_RTF |

cOdigo_concepto: CODIGO del concepto descrito, contenido en la base de datos. Será único para cada registro ~Q.

Este registro es de sustitución de la INFORMACION, no es de acumulación.

COdigo_seccion_pliego: DEFINICION del CODIGO asociado a cada pliego. Corresponde al definido en el registro de cabecera de pliego ~L.

CODIGO_PARRAFO: CODIGO del texto asociado a cada sección del pliego.

ABREV_AMBITO: Identificador del ámbito geográfico de la sección del pliego. Se define en un registro propio.

TEXTO_PARRAFO: Texto que define el contenido de los pliegos que se asocian a un concepto y se identifica con CODIGO_PARRAFO.

TEXTO_PARRAFO_RTF: Texto que define el contenido de los pliegos que se asocian a un concepto y se identifica con CODIGO_PARRAFO, con formato RTF, de forma opcional, siendo siempre obligatorio el campo TEXTO_PARRAFO.

REGISTRO TIPO AMBITO GEOGRAFICO

Establece el ámbito geográfico correspondiente a los Pliegos de Condiciones asociados a la Base de Datos. No necesariamente deberá corresponder al campo CABECERA definido en el registro ~V.

~W | { ABREV_AMBITO \ AMBITO \ } |

ABREV_AMBITO: Nombre abreviado que identifica el territorio geográfico al que se refiere. (Comunidad Autónoma, Provincia, Región, Comarca, Localidad, etc.). El identificador < * > (ASCII - 42) indica AMBITO_GENERAL, y representa todo el territorio nacional.

AMBITO: Nombre completo del territorio geográfico.

Existe una relación de abreviaturas recomendadas, elaborada por la Asociación de Redactores de Bases de Datos de CONSTRUCCION.

REGISTRO TIPO INFORMACION GRAFICA.

~G | CODIGO_CONCEPTO | { ARCHIVO_GRAFICO. EXT \ } |

CODIGO_CONCEPTO: CODIGO del concepto descrito en la base de datos y contenido en ella.

ARCHIVO_GRAFICO. EXT: Nombre del archivo que contiene la INFORMACION gráfica. Se usarán como referencia programas estandarizados de uso general, para chequear y verificar el contenido del fichero. Estos programas serán:

Ficheros tipo ráster: Extensión .BMP, .PCX : Windows 3.1

Ficheros vectoriales: Extensión .WMF : Windows 3.1

Extensión .DXF : Autocad 12 Windows

REGISTRO TIPO ENTIDAD.

Define las entidades suministradoras de documentación técnica, tarifas de precios y especificaciones de los conceptos contenidos en la Base de Datos.

~E | CODIGO_ENTIDAD | RESUMEN | NOMBRE | { TIPO \ SUBNOMBRE

\ DIRECCIÓN \ CP \ LOCALIDAD \ PROVINCIA \ PAIS \ { TELEFONO; }

\ { FAX; } \ { PERSONA_CONTACTO; } \ } |

CODIGO_ENTIDAD: CODIGO del SCc que define a la entidad (empresa, organismo, etc.).

RESUMEN: Nombre abreviado de la entidad

NOMBRE: Nombre completo de la entidad.

TIPO: Se definen los siguientes:

C central.

D delegación.

R representante.

SUBNOMBRE: Nombre de la delegación o representante en caso de que sea distinto de la central. Normalmente estará vacío.

DIRECCIÓN \ CP \ LOCALIDAD \ PROVINCIA \ PAIS: Dirección postal de la entidad con todos sus datos, existiendo una dirección por cada subcampo tipo, de forma ordenada y secuencial.

TELEFONO: Números de teléfono de la entidad, de forma ordenada y secuencial respecto al subcampo tipo, separados con el carácter < ; > (ASCII-59). Se indicará con nueve caracteres numéricos, incluido el prefijo de la provincia.

FAX: Números de fax de la entidad, con las mismas especificaciones que el campo anterior.

PERSONA_CONTACTO: Nombre de las personas de contacto con la entidad y cargo que desempeña, podrá haber varias asociadas a cada subcampo tipo, de forma que estén separadas por el carácter ASCII-59.

REGISTRO TIPO RELACIÓN COMERCIAL

Este registro establece los vínculos entre los conceptos de una Base de Datos General (BDG) con los productos comerciales de una Base de Datos Específica (BDE), y/o viceversa.

Así una Base de Datos (BD) podrá contener CONCEPTOS genéricos de una BDG, CONCEPTOS referentes a productos comerciales de una BDE, o ambas a la vez.

~O | CODIGO_RAIZ_BD # CODIGO_CONCEPTO | |

{ CODIGO_ARCHIVO \ CODIGO_ENTIDAD # CODIGO_CONCEPTO \ } |

CODIGO_RAIZ_BD # CODIGO_CONCEPTO: Identificador de un concepto de una BD, donde:

. CODIGO_RAIZ_BD: Se refiere a la identificación del CODIGO de la entidad que elabora la BD. Este CODIGO debe ser facilitado por la entidad que elabora la BD, para evitar ambigüedades.

. CODIGO_CONCEPTO: Se refiere a un concepto que pertenece a CODIGO_RAIZ_BD, y empleado por ésta en su sistema de clasificación por codificación.

CODIGO_ARCHIVO: Se refiere al nombre del archivo que, de existir, indica el lugar donde se encuentra la INFORMACION referente a CODIGO_ENTIDAD # CODIGO_CONCEPTO. Sin embargo si dicho CODIGO_ARCHIVO no existe, entonces indica que CODIGO_ENTIDAD # CODIGO_CONCEPTO se encuentra en la misma BD.

CODIGO_ENTIDAD # CODIGO_CONCEPTO: Identificador de un concepto de una BD, donde:

. CODIGO_ENTIDAD: Se refiere a la identificación del CODIGO de la entidad a la que se le asocia INFORMACION. Este CODIGO debe ser facilitado por la entidad que elabora la BD, de acuerdo con su sistema de clasificación, para evitar ambigüedades.

. CODIGO_CONCEPTO: Se refiere a un concepto que pertenece a CODIGO_ENTIDAD, y empleado por la entidad que elabora la BD en su sistema de clasificación por codificación.

Cuando CODIGO_CONCEPTO se refiera a un producto comercial, dicho CODIGO deberá ser facilitado por el fabricante, y no podrá coincidir nunca con la designación de CODIGO_RAIZ_BD, CODIGO_ENTIDAD o CODIGO_CONCEPTO, cuando éste se refiere a un concepto genérico. Al tener dicho producto comercial un tratamiento de CONCEPTO, éste puede utilizar todos los registros existentes en el formato para especificar su INFORMACION asociada (precio, INFORMACION gráfica, etc.).

REGISTRO TIPO INFORMACION TECNICA

Este registro contiene la ESPECIFICACION de otros datos referentes al concepto, como por ejemplo, peso específico o nominal, características físicas, cuantías geométricas, propiedades físico-mecánicas, etc.

Estos datos podrían emplearse en otras utilidades, como el cálculo de los coeficientes de transmisión térmica, aislamiento acústico, etc.

El registro tipo INFORMACION Técnica puede adoptar dos formas:

Si el primer campo está vacío, éste sirve como diccionario de términos de INFORMACION Técnica a los cuales se les podrá asociar una descripción y una unidad de medida.

~X | | { CODIGO_IT \ DESCRIPCION_IT \ UM \ } |

Si el primer campo identifica a un concepto, la INFORMACION que se especificará a continuación serán la/las parejas de términos de INFORMACION técnica con sus respectivos valores.

~X | CODIGO_CONCEPTO | { CODIGO_IT \ VALOR_IT \ } |

CODIGO_IT: CODIGO de la INFORMACION Técnica descrita.

DESCRIPCION_IT: Texto descriptivo de la INFORMACION Técnica, sin limitación de tamaño.

UM: En el caso que los valores de la INFORMACION Técnica sean valores numéricos, se indicará su Unidad de Medida, de acuerdo con el Sistema Internacional de Unidades de Medida.

CODIGO_CONCEPTO: CODIGO del concepto descrito, contenido en la base de datos. Será único para cada registro ~X.

VALOR_IT: Valor alfabético o numérico de la INFORMACION Técnica.

REGISTRO TIPO COEFICIENTES.

Indica el número de decimales en cada campo numérico.

~K | DN \ DD \ DS \ DR \ DI \ DP \ DC \ DM | CI |

DN : Decimales del campo número de partes iguales de la hoja de mediciones.

DD : Decimales de Dimensiones de las tres magnitudes de la hoja de mediciones.

DS : Decimales de la línea de subtotal o total de mediciones

DR : Decimales de rendimiento y factor en una descomposición.

DI : Decimales del importe resultante de multiplicar rendimiento x precio

del concepto.

DP : Decimales del importe resultante del sumatorio de los costes directos del concepto.

DC : Decimales del importe total del concepto. (CD+CI).

DM : Decimales del importe resultante de multiplicar la medición total del concepto por

su precio.

CI : Porcentaje de costes indirectos.

REGISTRO TIPO MEDICIONES

En este registro figuran las mediciones (cantidades), en que interviene un concepto de un presupuesto en la descomposición de otro de mayor rango.

En el intercambio de archivos de presupuestos, deberá figurar siempre este registro, exista o no desglose de mediciones.

Cuando se trate de intercambiar una relación de registros ~M que recogen un listado de mediciones no estructurado, no es necesario la disposición de un CODIGO raíz ni los registros ~D complementarios. El operador indicará en estos casos cual es el destino de la medición.

~M | [CODIGO_PADRE \ ] CODIGO_HIJO | { POSICION \ } | MEDICION_TOTAL

| { TIPO \ COMENTARIO \ UNIDADES \ LONGITUD \ LATITUD \ ALTURA \ } |

CODIGO_PADRE: CODIGO del concepto padre o concepto descompuesto del presupuesto.

CODIGO_HIJO: CODIGO del concepto hijo o concepto de la línea de descomposición.

Este campo es opcional en el caso de intercambiar mediciones no estructuradas, es decir, que no pertenecen a la estructura general y completa de un presupuesto.

POSICION: Posición del CONCEPTO_HIJO en la descomposición del CONCEPTO_PADRE, este dato permite identificar la medición cuando la descomposición del concepto padre incluye varios conceptos hijo con el mismo CODIGO, la numeración de las posiciones comenzará con el 1.

El campo POSICION deberá especificarse siempre en intercambio de presupuestos cuando éste sea completo y estructurado, e indicará el camino completo de la medición descrita en la estructura del archivo. Por ejemplo 3 \ 5 \ 2, indicará la medición correspondiente al capítulo 3 del archivo; subcapítulo 5 del capítulo 3; y partida 2 del subcapítulo 5. En mediciones no estructuradas este campo es opcional.

MEDICION_TOTAL: Debe coincidir con el rendimiento del registro tipo ’~D’ correspondiente.

Incorpora el sumatorio del producto de unidades, longitud, latitud y altura o el resultado de expresiones de cada línea, al leer este registro se recalculará este valor.

TIPO: Indica el tipo de línea de medición de que se trate. Usualmente este subcampo estará vacío. Los tipos establecidos en esta VERSION son:

’1’: Subtotal parcial: En esta línea aparecerá el subtotal de las líneas anteriores desde el último subtotal hasta la línea inmediatamente anterior a ésta.

’2’: Subtotal acumulado: En esta línea aparecerá el subtotal de todas las líneas anteriores desde la primera hasta la línea inmediatamente anterior a ésta.

’3’: Expresión: Indicará que en el subcampo COMENTARIO aparecerá una expresión algebraica a evaluar. Se podrán utilizar los operadores ’(’, ’)’, ’+’, ’-’, ’*’, ’/’ y ’^’; las variables ’a’, ’b’, ’c’ y ’d’ (que tendrán por valor las cantidades introducidas en los subcampos UNIDADES, LONGITUD, LATITUD y ALTURA respectivamente); y la constante ’p’ para el valor Pi=3.1415926. Si la expresión utiliza alguna variable, ésta será válida hasta la siguiente línea de medición en la que se defina otra expresión.

COMENTARIO: Texto en la línea de medición. Podrá ser un comentario o una expresión algebraica.

UNIDADES, LONGITUD, LATITUD, ALTURA: Cuatro número reales con las mediciones. Si alguna magnitud no existe se dejará este campo vacío.

REGISTRO TIPO AÑADIR MEDICIONES

Igual que el registro tipo ~M pero añade las líneas de medición de este registro a las ya existentes en vez de substituir toda la medición como hace en aquel.

~N | [CODIGO_PADRE \ ] CODIGO_HIJO | { POSICION \ } |MEDICION

| {TIPO \ COMENTARIO \ UNIDADES \ LONGITUD \ LATITUD \ ALTURA \ } |

REGISTRO TIPO CLAVES

Este registro establece la relación entre CODIGOs y descriptores del tesauro, para permitir la búsqueda de conceptos mediante términos clave.

~A | CODIGO_CONCEPTO | { CLAVE_TESAURO \ } |

CODIGO_CONCEPTO: CODIGO del concepto descrito en la base de datos y contenido en ella.

CLAVE_TESAURO: Términos clave relacionados con el concepto. Los términos compuestos (hormigón armado, cartón-yeso, mortero mixto) se identificarán unidos mediante < _ > (ASCII - 95), (hormigón_armado, cartón_yeso, mortero_mixto...). No está permitido el empleo del espacio en blanco.

REGISTRO TIPO CAMBIO DE CODIGO

Con este registro se posibilita el cambio o anulación de los CODIGOs de los conceptos, única unidad de INFORMACION que no se podía modificar con los registros definidos anteriormente.

~B | CODIGO_CONCEPTO | CODIGO_NUEVO |

CODIGO_CONCEPTO: CODIGO del concepto a cambiar o anular. Debe existir y pertenece a un concepto contenido en la BD

CODIGO_NUEVO: Nuevo CODIGO para CODIGO_CONCEPTO, no debe existir previamente. Si este campo está vacío, se entiende que hay que eliminar CODIGO_CONCEPTO.

FORMATO FIEBDC-3. RESUMEN.

~V | PROPIEDAD_ARCHIVO | VERSION_FORMATO \ DDMMAA | PROGRAMA_EMISION |

[CABECERA] \ { ROTULO_IDENTIFICACION \ } | JUEGO_CARACTERES |

~C | {CODIGO \ } | UNIDAD | RESUMEN | { PRECIO \ } | { FECHA \ } | TIPO |

~D | CODIGO_PADRE | { CODIGO_HIJO \ FACTOR \ RENDIMIENTO \ } |

~Y | CODIGO_PADRE | { CODIGO_HIJO \ FACTOR \ RENDIMIENTO \ } |

~T | CODIGO_CONCEPTO | TEXTO_DESCRIPTIVO |

~P | CODIGO_FAMILIA | DESCRIPCION_PARAMETRICA |
~P | | DESCRIPCION_PARAMETRICA | NOMBRE.DLL |

~L | | { CODIGO_SECCION_PLIEGO \ ROTULO_SECCION_PLIEGO \ } |

~L | CODIGO_CONCEPTO | {CODIGO_SECCION_PLIEGO \TEXTO_SECCION_PLIEGO \ } |

~Q | { CODIGO_CONCEPTO \ } | { CODIGO_SECCION_PLIEGO \ CODIGO_PARRAFO
\ { ABREV_AMBITO; } \ } |

~J | CODIGO_PARRAFO | TEXTO_PARRAFO | TEXTO_PARRAFO_RTF |

~W | { ABREV_AMBITO \ AMBITO \ } |

~G | CODIGO_CONCEPTO | { ARCHIVO_GRAFICO. EXT \ } |

~E | CODIGO_ENTIDAD | RESUMEN | NOMBRE | { TIPO \ SUBNOMBRE
\ DIRECCIÓN \ CP \ LOCALIDAD \ PROVINCIA \ PAIS \ { TELEFONO; }
\ { FAX; } \ { PERSONA_CONTACTO; } \ } |

~O | CODIGO_RAIZ_BD # CODIGO_CONCEPTO | | { CODIGO_ARCHIVO \
CODIGO_ENTIDAD # CODIGO_CONCEPTO \ } |

~X | | { CODIGO_IT \ DESCRIPCION_IT \ UM \ } |
~X | CODIGO_CONCEPTO | { CODIGO_IT \ VALOR_IT \ } |

~K | DN \ DD \ DS \ DR \ DI \ DP \ DC \ DM | CIE |

~M | [CODIGO_PADRE \ ] CODIGO_HIJO | { POSICION \ } | MEDICION

| { TIPO \ COMENTARIO \ UNIDADES \ LONGITUD \ LATITUD \ ALTURA \ } |

~N | [CODIGO_PADRE \ ] CODIGO_HIJO | { POSICION \ } | MEDICION

| { TIPO \ COMENTARIO \ UNIDADES \ LONGITUD \ LATITUD \ ALTURA \ } |

~A | CODIGO_CONCEPTO | { CLAVE_TESAURO \ } |

~B | CODIGO_CONCEPTO | CODIGO_NUEVO |

Anexo 1. Cambios respecto a FIEBDC-3/95.

A continuación se indica la lista de apartados y registros del FIEBDC-3/95 afectados por ampliaciones y/o modificaciones:

1. FORMATO FIEBDC-3. ESPECIFICACION.
2. ~V. Registro tipo PROPIEDAD Y VERSION.
3. ~C. Registro tipo CONCEPTO.
4. ~P. Registro tipo DESCRIPCION PARAMETRICA.
5. ~O. Registro tipo INFORMACION COMERCIAL, que pasa a denominarse RELACION COMERCIAL.
6. Se añade el registro ~X. Registro tipo INFORMACION TÉCNICA.
7. Se añade el nuevo tratamiento de DESCRIPCIÓN PARAMÉTRICA: API ESTANDAR para descripciones paramétricas compiladas en DLL (Anexo 3).

Anexo 2. DESCRIPCIÓN PARAMÉTRICA: Formato ESTANDAR.

Un concepto paramétrico es el que define su CODIGO, resumen, texto, pliego, descomposición e INFORMACION comercial de forma paramétrica, esto es, de una forma variable mediante tablas y expresiones aritméticas y lógicas función de parámetros.

En la descripción paramétrica se encuentran las siguientes sentencias:

Se definen las variables:

%A %B %C %D Parámetros seleccionados del concepto, de "a" a "z" ~ 1 a 26.

%O %P %Q %R Parámetros seleccionados de la obra, de "a" a "z" ~ 1 a 26.

%E Variable que define las condiciones de error.

$A $B $C $D Textos de los parámetros seleccionados del concepto.

$O $P $Q $R Textos de los parámetros seleccionados de la obra.

$E Variable que define los textos de error.

De forma equivalente las variables %O a %R y $O a $R tomarían el valor correspondiente a los valores de los parámetros generales de la obra.

Cualquier variable de la ’A’ a la ’Z’ tanto numérica (%) como alfanumérica ($) se puede definir o redefinir con cualquier número de dimensiones para ser utilizada posteriormente en expresiones.

Se definen las constantes de la ’a’ a la ’z’ con los valores numéricos del 1 al 26 respectivamente, para permitir referenciar los parámetros de forma nemotécnica. Para la utilización de otro tipo de caracteres, se determinará en el texto de la opción del parámetro seleccionado el carácter de sustitución que se desea utilizar, anteponiéndole un carácter especial ’ ! ’ .Si dicho carácter no existe la sustitución se realiza relacionando el carácter con la posición que ocupa.

Ejemplo: PBPO.2$ M3 Hormigón $B $A

\ CONSISTENCIA \ plástica \ fluida \ blanda \

\ RESISTENCIA \ H-125 \ H-150 \ H-175 \ H-200 \

El derivado PBPO.2aa sería: M3 Hormigón H-125 plástica

Con el carácter especial:

\ CONSISTENCIA \ !p plástica \ !f fluida \ !b blanda \

\ RESISTENCIA \ !2 H-125 \ !5 H-150 \ !7 H-175 \ !0 H-200 \

El mismo derivado sería: PBPO.2p2 M3 Hormigón H-125 plástica.

Las variables numéricas deben permitir valores reales en coma flotante de doble precisión (64bits) y las variables alfanuméricas deben poder almacenar textos de cualquier tamaño.

Cualquier variable puede definirse, en la misma asignación, con cualquier número y tamaño de dimensiones (hasta 4), en la DEFINICION de dimensiones tendrán que hacerse explícitas todas las dimensiones.

%U=..... # define una variable con un dato numérico

$X(8)=..... # define una lista de 8 datos alfanuméricos

%V(3,4)= ..... # define una tabla con 3 filas y 4 columnas de datos n.

Las variables %E y $E son especiales para devolver errores producidos por selecciones de parámetros no coherentes. En una evaluación secuencial de expresiones, si en una expresión la variable %E adopta un valor distinto de 0, ha habido algún error, se interrumpe la evaluación de expresiones y se presenta el contenido de la variable $E donde se almacena el texto del error producido.

Pueden haber múltiples asignaciones de %E, cada una de ellas precedida de su correspondiente texto de error, asignación de $E.

Las constantes alfanuméricas se definirán entre comillas ($I="incluida parte proporcional").

En la descripción paramétrica podemos encontrar los siguientes tipos de sentencias:

SENTENCIA DE ROTULOS DE PARAMETRO:

\ <ROTULO del parámetro> \ { <ROTULO de la opción> \ }

Los parámetros definidos, hasta 4, se irán asignando a las variables ABCD en el orden que se encuentren.

SENTENCIA DE ASIGNACION NUMERICA:

<variable numerica> = <expresión numérica>

SENTENCIA DE ASIGNACION ALFANUMERICA:

<variable alfanumerica> = <expresión alfanumérica>

SENTENCIA DE RENDIMIENTO:(CONCEPTOS DESCOMPUESTOS)

<texto de sustitución de CODIGO> : <expresión numérica> [: <exp.num.>] Se pueden definir uno u opcionalmente dos rendimientos, el defecto del rendimiento opcional es 1.

SENTENCIA DE MEDIOS AUXILIARES:

%: <expresión numérica> (en tanto por cien)

%%: <expresión numérica> (en tanto por uno)

SENTENCIA DE PRECIO:(CONCEPTOS SIMPLES) :: <expresión numérica>

En caso de figurar conjuntamente un juego de sentencias de rendimiento a modo de descomposición y una sentencia de precio, tendrá prioridad la sentencia de precio, ignorando las sentencias de rendimiento.

SENTENCIA DE COMENTARIO:

\ COMENTARIO \ ó \ C \ <texto del comentario> \

Si existe texto de comentario, se presentará como ayuda a la selección de parámetros junto a las opciones de éstos.

SENTENCIA DE SUSTITUCION:

\ RESUMEN \ ó \ R \ <texto de sustitución del texto resumido> \

\ TEXTO \ ó \ T \ <texto de sustitución del texto descriptivo> \

\ PLIEGO \ ó \ P \ { <texto de sustitución de pliego> \ }

\ CLAVES \ ó \ K \ { <texto de sustitución de clave> \ }

\ COMERCIAL \ ó \ F \ { <texto de sustitución de CODIGO>\ <expresión_numérica>\}

Se considera que una sentencia continua en la línea siguiente si:

. Acaba en un operador

. Acaba sin cerrar comillas ’"’

. Comienza con ’\’ y no acaba con ’\’

<constantes> PI, números, "texto" ...

<funciones> ABS( ), INT( ), SQRT( ) ...

<variables> [$%] [A-Z] [(dimensión{,dimensión})]

<expresión numérica>:

Son aquellas que dan como resultado un número en función de constantes y variables numéricas, expresiones lógicas, funciones y operadores.

por ejemplo: %I= %A + 3.17*(1+%B) + ABS(%P+3.15*%Q)/12000

<expresión alfanumérica>:

Son aquellas que dan como resultada un texto en función de constantes y variables alfanuméricas, operadores y funciones numéricas.

Una expresión alfanumérica puede incluir expresiones lógicas.

por ejemplo: $I="parte proporcional"+" de perdidas"*(%A>a)

añadir " de perdidas" a $I si el valor actual de %A es mayor que <a> ó 1.

<expresiones lógicas>:

Son aquellas que dan como resultado VERDADERO o FALSO. En expresiones numéricas el verdadero se considera como 1 y el falso como 0, en alfanuméricas el falso se considera suprimir texto.

%I = 323*(%A=a) + 345*(%A=b) + 1523*($I=$A & $J=$B)

$I = "blanco"*(%C=c) + "negro"*(%C=d)

<texto de sustitución>:

En los textos de sustitución la INFORMACION es un texto constante (sin comillas) con variables embebidas en él. Se consideran variables los caracteres $ y % inmediatamente seguidos por una letra de la A a la Z.

En los textos de sustitución, las variables alfanuméricas se sustituyen por sus contenido de texto correspondiente, las numéricas se sustituyen por las constantes de la "a" a la "z" correspondientes al valor numérico de su contenido.

En la expresión del rendimiento, la primera parte de la sentencia es un texto de sustitución que una vez sustituidas las variables será el CODIGO del concepto al que le corresponde la expresión numérica de la segunda parte de la expresión como rendimiento. Si el resultado es 0, se ignora la sentencia y no se considera ese componente o línea de descomposición.

CONVENIOS DE NOTACION (EBNF)

[a] Indica nada o "a"

{a} Indica cero o más ocurrencias de "a"

[a-b] Indica cualquier valor desde "a" a "b" inclusivas

[abc] Indica cualquiera de los valores a, b ó c

<abc> Indica descripción informal

abc Indica símbolo terminal

%[A-Z] Variable numérica

$[A-Z] Variable alfanumérica

Variables predefinidas:

[%$][ABCD] Parámetros del concepto

[%$][OPQR] Parámetros de la obra

[%$]E Variable especial para reportar errores

[%$][A-Z][(dim{,dim})] Variables definibles

Comentarios (el texto comprendido entre este carácter y el siguiente final de línea exclusive, no se tiene en cuenta)

, Separador de datos

: DEFINICION de rendimiento

:: DEFINICION de precio

%: DEFINICION de medios auxiliares en tanto por cien

%%: DEFINICION de medios auxiliares en tanto por uno

OPERADORES NUMERICOS (De menor a mayor precedencia)

+ Suma

- Resta

* Multiplicación

/ División

^ Operador elevado a

OPERADORES LOGICOS (De menor a mayor precedencia)

@ Operador lógico O

& Operador lógico Y

< Menor

> Mayor

<= Menor o igual

>= Mayor o igual

= Igual

<> Diferente

! Operador lógico NO

FUNCIONES -- VALOR RETORNADO:

ABS(n) Valor absoluto de "n"

INT(n) Parte entera de "n"

ROUND(n,d) Redondeo de "n" a "d" decimales

SIN(n) Seno (grados sexagesimales)

COS(n) Coseno (grados sexagesimales)

TAN(n) Tangente (grados sexagesimales)

ASIN(n) Arco seno (gs)

ACOS(n) Arco coseno (gs)

ATAN(n) Arco tangente (gs)

ATAN2(x,y) Arco tangente con dos parámetros "x" e "y"

SQRT(n) Raíz cuadrada de "n"

ATOF(a) ConVERSION de alfanumérico "a" a numérico

FTOA(n) ConVERSION de numérico "n" a alfanumérico

Cada instrucción irá en distinta línea, a menos que la instrucción acabe en un operador en cuyo caso se considera que sigue en la siguiente línea.

Si una línea acaba sin haber cerrado las comillas '"' o delimitador '\', se considerará que sigue en la línea siguiente. Los caracteres fin de línea (ASCII-13 + ASCII-10) contenidos en las descripciones paramétricas se mantendrán al reformatear.

CONTROL DE ERRORES DE SELECCION.

Es frecuente encontrar un gran número de combinaciones de parámetros posibles pero tener pocas de ellas resueltas. Para evitar que el operador del sistema pruebe distintas combinaciones de parámetros consiguiendo en todas ellas un mensaje de error, el sistema debe ser capaz de guiarle en la selección de combinaciones correctas.

Cada vez que el operador define un parámetro, el sistema evaluará todas las sentencias posibles y en las sentencias del tipo: %E= ...., función de parámetros

Si todos los parámetros intervinientes en la expresión son conocidos, se evaluará ésta y si el resultado fuera de ERROR se presentará la previa DEFINICION de $E con el mensaje del error.

Si todos los parámetros menos uno son conocidos, se irá dando valores al parámetro desconocido y evaluando la expresión hasta recorrer todos los valores válidos del parámetro. De alguna forma, el sistema "marcará" los valores que producen ERROR del parámetro estudiado en la pantalla de selección, para ayudar al operador a seleccionar las combinaciones correctas.

Cada vez que se defina o redefina un parámetro el sistema actualizará todos los valores marcados en pantalla, por ejemplo pondrá en "medio brillo" los ROTULOs de las opciones cuya selección no sería compatible con los parámetros seleccionados previamente.

Este sistema de control de errores de selección es sencillo de implementar en cualquier soporte, pero obliga a los redactores de los descompuestos paramétricos a definir explícitamente las combinaciones de parámetros incorrectas, ya que con este método no se podrían encontrar combinaciones no permitidas cuando en la descomposición paramétrica se llama a otros descompuestos o precios paramétricos.

PROCEDIMIENTO DE LECTURA DE DESCRIPCIONES PARAMETRICAS.

Recorrer la descripción paramétrica ejecutando los siguientes pasos:

1. Eliminar desde el carácter ’#’ inclusive hasta el siguiente cambio de línea exclusive.

2. Cambiar tabuladores (9) por caracteres ’ ’ (32)

3. Eliminar caracteres ’ ’ (32) delante y detrás de los caracteres ’\’

4. Unir líneas, eliminando el fin de línea, en líneas que comienzan con ’\’ y no acaban con ’\’, que terminan con un operador y en la separación de datos de una variable matricial.

5. Eliminar todos los caracteres ’ ’ (32) en zonas no entrecomilladas ("...") o delimitadas (\...\)

6. Eliminar líneas vacías.

7. Leer y evaluar secuencialmente las sentencias de la forma:

Si la sentencia comienza con '\' leer el ROTULO hasta el siguiente '\', si el ROTULO es:

COMENTARIO ó C- Palabra o carácter reservado que identifica el siguiente ROTULO entre '\' como comentario a la selección de parámetros.

RESUMEN ó R- Palabra o carácter reservado que identifica el siguiente ROTULO entre '\' como el texto de sustitución del resumen del concepto.

TEXTO ó T- Palabra o carácter reservado que identifica el siguiente ROTULO entre '\' como el texto de sustitución del texto descriptivo del concepto.

PLIEGO ó P- Palabra o carácter reservado que identifica los siguientes ROTULOs entre '\' como los textos de sustitución de las distintas secciones del pliego.

CLAVES ó K- Palabra o carácter reservado que identifica los siguientes ROTULOs entre '\' como los textos de sustitución de los términos claves asociados al concepto.

COMERCIAL ó F - Palabra o carácter reservado que identifica los siguientes ROTULOs entre '\' como los textos de sustitución y tarifas de la INFORMACION comercial del concepto.

Cualquier otro ROTULO identificará el nombre del siguiente parámetro y los siguientes ROTULOs entre '\' como los ROTULOs de las opciones de dicho parámetro.

Si la sentencia comienza con '::' el resto de la misma debe ser una expresión numérica indicadora del Precio, sólo en familias de conceptos simples (sin descomposición) y sólo puede haber una sentencia de este tipo.

Si la sentencia comienza con '%:' el resto de la misma debe ser una expresión numérica indicadora del Porcentaje de Medios Auxiliares, sólo puede haber una sentencia de este tipo.

En otro caso si la sentencia contiene el carácter ':' la parte anterior a él es un texto de sustitución del CODIGO de una línea de descomposición y la posterior una expresión numérica, o dos separadas por ':', indicadoras de el o los rendimientos de dicha línea de descomposición.

En aquellos casos donde pueda aparecer un carácter ?%? seguido de un carácter alfabético que se considere como tal y no como una variable de sustitución, deberá emplearse ?%%?, para evitar la ambigüedad que se puede producir entre una variable numérica que deba ser sustituida, una sentencia de medio auxiliar o un texto.

El resto de sentencias deberán ser de asignación de la forma variable/s = expresión/es

RESUMEN DE TIPOS DE SENTENCIAS

Después de realizado el filtro descrito arriba, cada línea, tira de caracteres acabada en (ASCII-13)(ASCII-10), será una sentencia de alguno de los siguientes tipos:

{ \ ROTULO_parámetro \ { opción_parámetro \ } (13)(10) }

{ variable = expresión (13)(10) }

{ CODIGO : rendimiento [: rendimiento] (13)(10) }

[%: ó %%: medios_auxiliares (13)(10)] % (tanto por cien) %% (tanto por uno)

[:: expresión_precio (13)(10)]

[\ COMENTARIO \ ó \ C \ texto_comentario \ (13)(10)]

[\ RESUMEN \ ó \ R \ texto_resumen \ (13)(10)]

[\ TEXTO \ ó \ T \ texto_descriptivo \ (13)(10)]

[\ PLIEGO \ ó \ P \ { texto_faceta_pliego \ }(13)(10)]

[\ CLAVES \ ó \ K \ { término_clave \ }(13)(10)]

[\ COMERCIAL \ ó \ F \ { CODIGO_producto_comercial \ tarifa \ } (13)(10)]

Anexo 3. DESCRIPCIÓN PARAMÉTRICA: API ESTANDAR para descripciones paramétricas compiladas en DLL.

INTRODUCCION

Debido a la necesidad expuesta por los desarrolladores de bases de datos paramétricas de ampliar las posibilidades del lenguaje de descripción paramétrica, poder compilar éste por eficiencia y protección de datos y posibilitar la protección contra copia de bases de datos paramétricas, se establece la siguiente ESPECIFICACION.

En este documento se definen los componentes necesarios para el desarrollo de descripciones paramétricas en cualquier lenguaje de aplicaciones para Windows (C, C++, Pascal, Fortran, etc.) y sin ninguna limitación. Se incluye la DEFINICION de un API ESTANDAR en C, un ejemplo de base de datos en formato DLL de 32 bits desarrollado en C++ y un ejemplo de aplicación con la implementación del interfaz con el API en C, definidos ambos en Microsoft Visual C++. Se podría implementar el interfaz con el API para otros compiladores y lenguajes para acceder a las mismas DLL.

Es decir; es posible construir una base de datos que cumpla este API utilizando para ello cualquier lenguaje de programación que permita desarrollar librerías de enlace dinámico Windows (DLL). Asimismo, es posible construir un programa que lea cualquier base de datos de estas características utilizando lenguajes de aplicaciones para Windows.

ARCHIVOS QUE DEBE CONTENER UNA BASE DE DATOS

Una base de datos que se desee distribuir con las definiciones paramétricas compiladas en DLL, debe contener los siguientes archivos:

base.dll En este archivo, único para cada base de datos y de nombre cualquiera pero extensión ?..DLL?, se encuentran las funciones del API que la base de datos ofrece a las aplicaciones para que éstas obtengan la INFORMACION que contiene la base.

base.bc3 Archivo o archivos ASCII de la base de datos en formato FIEBDC-3/98. Los registros '~P de los conceptos cuya descripción paramétrica se acceda a través del archivo ?base.dll?, tendrán el campo ?DESCRIPCIÓN_PARAMÉTRICA? vacío. Ejemplo:

~P|ABCD12$| |

El registro '~P' correspondiente al paramétrico global, tendrá el campo ?DESCRIPCIÓN_PARAMÉTRICA? vacío, y tendrá un tercer campo con el nombre del archivo DLL en el que se encuentren las funciones del API de la base. Ejemplo:

~P| | | BASE.DLL |

La DEFINICION paramétrica de los conceptos implementados de esta forma podrá estar en el mismo archivo que las funciones del API (el archivo ?base.dll?) o situado en otro u otros archivos cualesquiera, conforme desee el desarrollador de la base de datos. Las aplicaciones sólo accederán a las funciones del API incluidas en el archivo ?base.dll?, y éstas serán las encargadas de acceder a la INFORMACION en la forma que el desarrollador de la base implemente.

DEFINICION DEL API: FIEBDC.H

Único archivo que define el ESTANDAR. En este archivo se define el API en C, que las descripciones paramétricas en DLL ofrecen a las aplicaciones. Este interfaz permite definiciones paramétricas de ilimitado número de parámetros e ilimitadas opciones por parámetro. Se soportan dos modelos de codificaciones:

1. Un modelo de codificación independiente de parámetros, en el que el CODIGO de un concepto paramétrico es completamente libre y el número de caracteres del CODIGO es independiente del número de parámetros.

2. Un modelo dependiente de los mismos. Es el modelo que definía FIEBDC-3/95 y en el que el CODIGO de un concepto paramétrico debe tener un símbolo ?$? en su séptima posición y en el que se asigna de la 'a' a la 'z' las opciones 0 a 25 de cada parámetro, ampliándose en esta VERSION con los rangos 'A' a 'Z' y '0' a '9' para que el número de opciones por parámetro en este modelo de codificación pase a 62 (de 0 a 61).

Para que los programas puedan determinar si una base de datos responde a uno u otro modelo, se ha definido la función BdcCodificacion(), que se especifica más adelante y que indica si el sistema de codificación usado en la base de datos es dependiente o independiente.

Si se adopta el primer modelo, no es posible averiguar ?a priori?, a partir de un CODIGO ?ABCDEFGHIJ? de concepto, si éste es un derivado paramétrico ni de que concepto paramétrico procede o con qué valores de sus

parámetros. Por ello, es establece el siguiente criterio de búsqueda:

1. Si el concepto existe con este CODIGO en la base, se escogerá dicho concepto.

2. En caso de no existir, se intentará localizarlo en la base de datos como perteneciente a un concepto paramétrico ?al estilo? FIEBDC-3/95. En el ejemplo, se intentará buscar el concepto paramétrico ?ABCDEF$? y pasarle los parámetros ?GHIJ? (que implica pasarle a sus cuatro parámetros valores ?31?, ?32?, ?33? y ?34? respectivamente).

3. En caso de no existir, se intentará localizarlo en la DLL. Si ésta posee un modelo de codificación dependiente, se utilizará el mismo criterio que en el punto anterior: en el ejemplo, buscar el concepto paramétrico ?ABCDEF$? y pasarle los parámetros ?GHIJ?. Si la base posee un modelo independiente, se utilizará la función ?BdcDecodifica()?, tal como se especifica más adelante.

4. Si no se cumplen ninguna de las condiciones anteriores, se supone que el concepto no existe en la base.

El archivo ?fiebdc.h? tiene el siguiente contenido:

```
/* FORMATO DE INTERCAMBIO ESTANDAR DE PARAMÉTRICOS EN DLL */
/* FIEBDC-3/98 */

#include <windows.h>

#ifndef FIEBDC_H
#define FIEBDC_H

#ifdef BASE /* definido si se desea construir la DLL */

/*****************************************************************************/
/* PARTE DEL ARCHIVO NECESARIA PARA LOS DESARROLLADORES DE BASE DE DATOS */
/*****************************************************************************/

/*****************************************************************************/
/* MACROS DEPENDIENTES DEL COMPILADOR */
#if defined (__BORLANDC__) /* Borland C++ */
#define EXPORTA FAR _export
#elif defined (_MSC_VER) /* Microsoft C */
#define EXPORTA
#else /* Otros */
#define EXPORTA
#endif
```

```
#ifdef __cplusplus
extern "C" {
#endif

/* 0 FUNCIONES GENERALES ************************************************/
LONG EXPORTA BdcCodificacion (VOID);

/* 1 FUNCIONES REFERENTES AL PARAMÉTRICO GLOBAL *************************/
/* 1.1 Accesibles en cualquier momento */
/* 1.1.1 Obtención de sus parámetros */
LONG EXPORTA BdcGloParNumero (VOID);
LONG EXPORTA BdcGloOpcNumero (LONG par);
LPCSTR EXPORTA BdcGloParRotulo (LONG par);
LPCSTR EXPORTA BdcGloOpcRotulo (LONG par, LONG opc);
/* 2.1.3 Mensajes / CODIGOs de error */
BOOL EXPORTA BdcGloError (LPCSTR *err);
/* 1.1.2 Asignación de opciones a los parámetros */
BOOL EXPORTA BdcGloCalcula (LPLONG opcl);

/* 2 FUNCIONES REFERENTES AL RESTO DE PARAMÉTRICOS **********************/
/* 2.1 Accesibles en cualquier momento */
/* 2.1.1 Lectura de un concepto paramétrico */
HANDLE EXPORTA BdcLee (LPCSTR cod);
/* 2.1.2 Lectura de un concepto paramétrico a partir del CODIGO del derivado */
HANDLE EXPORTA BdcDecodifica (LPCSTR cod, LPLONG opcl);
/* 2.1.3 Mensajes / CODIGOs de error */
BOOL EXPORTA BdcError (HANDLE h, LPCSTR *err);

/* 2.2 Accesibles después de BdcLee */
/* 2.2.1 Obtención de sus parámetros */
LONG EXPORTA BdcParNumero (HANDLE h);
LONG EXPORTA BdcOpcNumero (HANDLE h, LONG par);
LPCSTR EXPORTA BdcParRotulo (HANDLE h, LONG par);
LPCSTR EXPORTA BdcOpcRotulo (HANDLE h, LONG par, LONG opc);
/* 2.2.2 Obtención de un comentario */
LPCSTR EXPORTA BdcComentario (HANDLE h);
/* 2.2.3 Asignación de opciones de los parámetros y cálculo/chequeo del derivado */
BOOL EXPORTA BdcValida (HANDLE h, LPLONG opcl);
BOOL EXPORTA BdcCalcula (HANDLE h, LPLONG opcl);
```

```
/* 2.2.4 Liberación de memoria */
BOOL EXPORTA BdcCierra (HANDLE h);

/* 2.3 Accesibles después de BdcCalcula */
/* 2.3.1 Obtención del derivado paramétrico */
LONG EXPORTA BdcDesNumero (HANDLE h);
LPCSTR EXPORTA BdcDesCodigo (HANDLE h, LONG des);
BOOL EXPORTA BdcRendimiento (HANDLE h, LONG des, double FAR *ren);
BOOL EXPORTA BdcPrecio (HANDLE h, double FAR *pre);
LPCSTR EXPORTA BdcCodigo (HANDLE h);
LPCSTR EXPORTA BdcResumen (HANDLE h);
LPCSTR EXPORTA BdcTexto (HANDLE h);
LPCSTR EXPORTA BdcPliego (HANDLE h);
LPCSTR EXPORTA BdcClaves (HANDLE h);

#ifdef __cplusplus
}
#endif

#else /* BASE no definido */
/*****************************************************************************/
/* PARTE DEL ARCHIVO NECESARIA PARA LOS DESARROLLADORES DE PROGRAMAS */
/*****************************************************************************/

/*****************************************************************************/
/* MACROS DEPENDIENTES DEL COMPILADOR */
#if defined (__BORLANDC__) /* Borland C++ */
#define IMPORTA FAR _import
#elif defined (_MSC_VER) /* Microsoft C */
#define IMPORTA
#else /* Otros */
#define IMPORTA
#endif

/* FUNCIONES GENERALES ***************************************************/
typedef LONG (IMPORTA * BDCCODIFICACION) (VOID);
/* FUNCIONES REFERENTES AL PARAMÉTRICO GLOBAL ****************************/
typedef LONG (IMPORTA * BDCGLOPARNUMERO) (VOID);
typedef LONG (IMPORTA * BDCGLOOPCNUMERO) (LONG par);
```

```
typedef LPCSTR (IMPORTA * BDCGLOPARROTULO) (LONG par);
typedef LPCSTR (IMPORTA * BDCGLOOPCROTULO) (LONG par, LONG opc);
typedef BOOL (IMPORTA * BDCGLOERROR) (LPCSTR *err);
typedef BOOL (IMPORTA * BDCGLOCALCULA) (LPLONG opcl);

/* FUNCIONES REFERENTES AL RESTO DE PARAMÉTRICOS ************************/
typedef HANDLE (IMPORTA * BDCLEE) (LPCSTR cod);
typedef HANDLE (IMPORTA * BDCDECODIFICA) (LPCSTR cod, LPLONG opcl);
typedef BOOL (IMPORTA * BDCERROR) (HANDLE h, LPCSTR *err);
typedef LONG (IMPORTA * BDCPARNUMERO) (HANDLE h);
typedef LONG (IMPORTA * BDCOPCNUMERO) (HANDLE h, LONG par);
typedef LPCSTR (IMPORTA * BDCPARROTULO) (HANDLE h, LONG par);
typedef LPCSTR (IMPORTA * BDCOPCROTULO) (HANDLE h, LONG par, LONG opc);
typedef LPCSTR (IMPORTA * BDCCOMENTARIO) (HANDLE h);
typedef BOOL (IMPORTA * BDCVALIDA) (HANDLE h, LPLONG opcl);
typedef BOOL (IMPORTA * BDCCALCULA) (HANDLE h, LPLONG opcl);
typedef BOOL (IMPORTA * BDCCIERRA) (HANDLE h);
typedef LONG (IMPORTA * BDCDESNUMERO) (HANDLE h);
typedef LPCSTR (IMPORTA * BDCDESCODIGO) (HANDLE h, LONG des);
typedef BOOL (IMPORTA * BDCRENDIMIENTO) (HANDLE h, LONG des, double FAR *ren);
typedef BOOL (IMPORTA * BDCPRECIO) (HANDLE h, double FAR *pre);
typedef LPCSTR (IMPORTA * BDCCODIGO) (HANDLE h);
typedef LPCSTR (IMPORTA * BDCRESUMEN) (HANDLE h);
typedef LPCSTR (IMPORTA * BDCTEXTO) (HANDLE h);
typedef LPCSTR (IMPORTA * BDCPLIEGO) (HANDLE h);
typedef LPCSTR (IMPORTA * BDCCLAVES) (HANDLE h);

#endif /* de #ifdef BASE */

/*****************************************************************************/
/* PARTE COMÚN DEL ARCHIVO: CODIGOS DE LOS MENSAJES DE ERROR */
/* SE ALMACENAN COMO BITS DE UN LONG, DE MANERA QUE EXISTAN HASTA 32 */
/*****************************************************************************/

#define BDCERR_CORRECTO 0x0000 /* No hay error */
#define BDCERR_BASE_DATOS 0x0001 /* Existe un mensaje de error definido por */
/* el redactor de la base */
#define BDCERR_PARAMETRO 0x0002 /* Se pasó a BdcCalcula o BdcGloCalcula */
/* un parámetro inexistente */
```

```
#define BDCERR_OPCION 0x0004 /* Se pasó a BdcCalcula o BdcGloCalcula una */
/* opción inexistente */
#define BDCERR_MAX_OPCIONES 0x0008 /* Se definieron más de 62 opciones */
#define BDCERR_NO_LEIDO 0x0010 /* Se intentó calcular un concepto sin leer */
#define BDCERR_NO_CALCULADO 0x0020 /* Se intentó acceder a datos de un derivado */
/* no calculado */
#define BDCERR_DESCOMPOSICION 0x0040 /* Se intentó acceder a un elemento de */
/* la descomposición inexistente */
#define BDCERR_SIN_CODIGO 0x0080 /* No existe CODIGO definido */
#define BDCERR_SIN_MEMORIA 0x0100 /* Memoria insuficiente */
#define BDCERR_CONCEPTO_NULO 0x0200 /* Se pasó un HANDLE nulo */

#endif /* FIEBDC_H */
```

ESPECIFICACION DE LAS FUNCIONES DEL API

1. FUNCIONES GENERALES

LONG EXPORTA BdcCodificacion (VOID);

Propósito

Indica si la base de datos utiliza un modelo de codificación dependiente o independiente del número y valor de los parámetros.

Valor devuelto

Devolverá ?0? si la codificación sigue un modelo dependiente (al ?estilo? FIEBDC-3/95), y ?1? si sigue un modelo independiente.

2. FUNCIONES REFERENTES AL PARAMÉTRICO GLOBAL

2.1. Accesibles en cualquier momento

2.1.1. Obtención de sus parámetros

LONG EXPORTA BdcGloParNumero (VOID);

Propósito

Obtener el número de parámetros de concepto paramétrico global.

Valor devuelto

Devuelve el número de parámetros. En caso de error, la función devuelve -1. Para obtener más INFORMACION sobre el error producido, llame a la función BdcGloError().

LONG EXPORTA BdcGloOpcNumero (LONG par // número de parámetro del concepto);

Propósito

Obtiene el número de opciones de que consta el parámetro ?par?.

Parámetros

par: Número del parámetro. Debe ser un valor entre ?0? y ?n-1?, siendo ?n? el número de parámetros del concepto paramétrico global.

Valor devuelto

Devuelve el número de opciones del parámetro ?par?. En caso de producirse un error, devuelve -1. Para obtener más INFORMACION sobre el error producido, llame a la función BdcGloError().

LPCSTR EXPORTA BdcGloParRotulo (LONG par // número de parámetro del concepto);

Propósito

Obtiene el rótulo que identifica el parámetro ?par? del concepto.

Parámetros

par: Número del parámetro. Debe ser un valor entre ?0? y ?n-1?, siendo ?n? el número de parámetros del concepto paramétrico global.

Valor devuelto

Devuelve el rótulo que identifica el parámetro ?par? del concepto, como puntero constante ?far? a una cadena de caracteres. La propia función es responsable de asignar memoria al puntero. En caso de error, la función devuelve NULL. Para obtener más INFORMACION sobre el error producido, llame a la función BdcGloError().

LPCSTR EXPORTA BdcGloOpcRotulo (LONG par, // número de parámetro del concepto LONG opc // número de la opción del parámetro);

Propósito

Obtiene el rótulo que identifica la opción ?opc? del parámetro ?par? del concepto.

Parámetros

par: Número del parámetro. Debe ser un valor entre ?0? y ?n-1?, siendo ?n? el número de parámetros del concepto paramétrico global.

opc: Número del parámetro. Debe ser un valor entre ?0? y ?n-1?, siendo ?n? el número de opciones que posee el parámetro ?par? del concepto paramétrico global.

Valor devuelto

Devuelve el rótulo que identifica la opción ?opc? del parámetro ?par? del concepto, como puntero constante ?far? a una cadena de caracteres. La propia función es responsable de asignar memoria al puntero. En caso de error, la función devuelve NULL. Para obtener más INFORMACION sobre el error producido, llame a la función BdcGloError().

2.1.2. Mensajes / CODIGOs de error

BOOL EXPORTA BdcGloError (LPCSTR *err // mensaje de error devuelto);

Propósito

Obtiene el tipo de error producido. Una vez leído, se inicializa el CODIGO de error.

Parámetros

err: Puntero a un puntero constante ?far? a una cadena de caracteres. En él se almacena el mensaje de error referente al error producido. La función es responsable de asignar memoria al puntero. Si no existe un mensaje definido para el error existente, ?*err? apuntará a la cadena vacía "".

Valor devuelto

Devuelve el CODIGO de error producido. Vea al final el apartado ?CODIGOs de los mensajes de error? para más INFORMACION.

2.1.3. Asignación de opciones a los parámetros

BOOL EXPORTA BdcGloCalcula (LPLONG opcl, // lista de opciones de los parámetros);

Propósito

Asigna los valores de los parámetros del concepto paramétrico global..

Parámetros

opcl: Puntero a un array de LONGs con las opciones que se desea fijar a cada parámetro. Las opciones se numeran empezando por cero.

Valor devuelto

Devuelve ?0? si se ejecuta correctamente. En caso de error, devuelve ?-1?. Para obtener más INFORMACION sobre el error producido, llame a la función BdcGloError().

3. FUNCIONES REFERENTES AL RESTO DE PARAMÉTRICOS

3.1. Accesibles en cualquier momento

3.1.1. Lectura de un concepto paramétrico

HANDLE EXPORTA BdcLee (LPCSTR cod // CODIGO del concepto);

Propósito

Lee el concepto paramétrico identificado por ?cod?.

Parámetros

cod: Puntero constante ?far? a una cadena de caracteres con el CODIGO del concepto paramétrico a leer. Si se utiliza un modelo de codificación dependiente, se asume que dicho CODIGO tenga 7 caracteres y que el séptimo sea ?$?. Dentro del CODIGO, los caracteres pueden ser cualesquiera salvo el 0x00 (que indica el final del CODIGO).

Valor devuelto

Si la función encuentra el paramétrico, retorna un HANDLE distinto de cero. En caso de error, o si no existe el paramétrico, la función devuelve cero.

3.1.2. Lectura de un concepto paramétrico a partir del CODIGO completo del derivado

HANDLE EXPORTA BdcDecodifica (LPCSTR cod, // CODIGO completo del derivado paramétrico LPLONG opcl // puntero al espacio de memoria a); // rellenar con las opciones

Propósito

Lee el concepto paramétrico al que pertenece el concepto de CODIGO ?cod?. El HANDLE y las opciones ?opcl? devueltas se pueden utilizar directamente en una llamada a BdcCalcula().

Parámetros

cod: Puntero constante ?far? a una cadena de caracteres con el CODIGO del concepto del que se desea obtener el concepto paramétrico a la que pertenece. Dentro del CODIGO, los caracteres pueden ser cualesquiera salvo el 0x00 (que indica el final del CODIGO).

opcl: Puntero a un array de LONGs en el que la función devolverá las opciones a las que corresponda el derivado paramétrico. El array debe estar previamente dimensionado con al menos el número de parámetros del concepto. Las opciones se numeran empezando por cero.

Valor devuelto

Si la función encuentra el paramétrico, retorna un HANDLE distinto de cero. En caso de error, o si no existe niguna concepto paramétrico del que el concepto ?cod? es derivado, la función devuelve cero.

3.1.3. Mensajes / CODIGOs de error

BOOL EXPORTA BdcError (HANDLE h, // identificador del concepto LPCSTR *err // mensaje de error devuelto);

Propósito

Obtiene tipo de error producido.

Parámetros

h: Identificador (HANDLE) del concepto, que debe ser obtenido en una llamada anterior a la función BdcLee().

err: Puntero a un puntero constante ?far? a una cadena de caracteres. En él se almacena el mensaje de error referente al error producido. La función es responsable de asignar memoria al puntero. Si no existe un mensaje definido para el error existente, ?*err? apuntará a la cadena vacía "".

Valor devuelto

Devuelve el CODIGO de error producido. Vea al final el apartado ?CODIGOs de los mensajes de error? para más INFORMACION.

3.2. Accesibles después de BdcLee

3.2.1. Obtención de sus parámetros

LONG EXPORTA BdcParNumero (HANDLE h // identificador del concepto);

Propósito

Obtiene el número de parámetros de concepto paramétrico.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

Valor devuelto

Devuelve el número de parámetros. En caso de error, la función devuelve -1. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

LONG EXPORTA BdcOpcNumero (HANDLE h, // identificador del concepto LONG par // número de parámetro del

concepto);

Propósito

Obtiene el número de opciones de que consta el parámetro ?par?.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

par: Número del parámetro. Debe ser un valor entre ?0? y ?n-1?, siendo ?n? el número de parámetros del concepto.

Valor devuelto

Devuelve el número de opciones del parámetro ?par?. En caso de producirse un error, la función devuelve -1. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

LPCSTR EXPORTA BdcParRotulo (HANDLE h, // identificador del concepto LONG par // número de parámetro del concepto);

Propósito

Obtiene el rótulo que identifica el parámetro ?par? del concepto.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

par: Número del parámetro. Debe ser un valor entre ?0? y ?n-1?, siendo ?n? el número de parámetros del concepto.

Valor devuelto

Devuelve el rótulo que identifica el parámetro ?par? del concepto, como puntero constante ?far? a una cadena de caracteres. La propia función es responsable de asignar memoria al puntero. En caso de error, la función devuelve NULL. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

LPCSTR EXPORTA BdcOpcRotulo (HANDLE h, // identificador del concepto LONG par, // número de parámetro del concepto LONG opc // número de la opción del parámetro);

Propósito

Obtiene rótulo que identifica la opción ?opc? del parámetro ?par? del concepto.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

par: Número del parámetro. Debe ser un valor entre ?0? y ?n-1?, siendo ?n? el número de parámetros del concepto.

opc: Número del parámetro. Debe ser un valor entre ?0? y ?n-1?, siendo ?n? el número de opciones que posee el parámetro ?par? del concepto.

Valor devuelto

Devuelve el rótulo que identifica la opción ?opc? del parámetro ?par? del concepto, como puntero constante ?far? a una cadena de caracteres. La propia función es responsable de asignar memoria al puntero. En caso de error, la función devuelve NULL. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

3.2.2. Obtención de un comentario

LPCSTR EXPORTA BdcComentario (HANDLE h, // identificador del concepto);

Propósito

Obtiene un texto de comentario del concepto paramétrico.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

Valor devuelto

Devuelve el comentario del concepto, como puntero constante ?far? a una cadena de caracteres. La propia función es responsable de asignar memoria al puntero. En caso de error, la función devuelve NULL. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

3.2.3. Asignación de opciones de los parámetros y validación o cálculo del derivado

BOOL EXPORTA BdcValida (HANDLE h, // identificador del concepto LPLONG opcl, // lista de opciones de los parámetros);

Propósito

Averigua si una determinada combinación paramétrica es correcta o no.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

opcl: Puntero a un array de LONGs con las opciones que se desea fijar a cada parámetro. Las opciones se numeran empezando por cero.

Valor devuelto

Devuelve ?0? si la combinación es correcta. En caso contrario, devuelve ?-1?. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

BOOL EXPORTA BdcCalcula (HANDLE h, // identificador del concepto LPLONG opcl, // lista de opciones de los parámetros);

Propósito

Calcula los datos correspondientes a un derivado paramétrico.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

opcl: Puntero a un array de LONGs con las opciones que se desea fijar a cada parámetro. Las opciones se numeran empezando por cero.

Valor devuelto

Devuelve ?0? si se ejecuta correctamente. En caso de error, o de que la combinación no sea correcta, devuelve ?-1?. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

3.2.4. Liberación de memoria

BOOL EXPORTA BdcCierra (HANDLE h // identificador del concepto);

Propósito

Cierra el concepto paramétrico y libera la memoria asignada.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

Valor devuelto

Devuelve ?0? si realiza la operación correctamente. En caso de error, la función devuelve ?-1?. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

3.3. Accesibles después de BdcCalcula

3.3.1. Obtención del derivado paramétrico

LONG EXPORTA BdcDesNumero (HANDLE h // identificador del concepto);

Propósito

Obtiene el número de conceptos en los que se descompone el derivado paramétrico. Es posible que un mismo concepto paramétrico posea derivados simples y compuestos.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

Valor devuelto

Devuelve el número de elementos de su descomposición. En caso de error, la función devuelve ?-1?. Un valor de cero

indicará que el concepto no tiene descomposición. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

LPCSTR EXPORTA BdcDesCodigo (HANDLE h, // identificador del concepto LONG des // número del elemento de la descomposición);

Propósito

Obtiene el CODIGO del elemento número ?des? en el que se descompone el derivado paramétrico.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

des: Número del elemento de la descomposición del concepto. Los elementos se numeran empezando por cero.

Valor devuelto

Devuelve el CODIGO del elemento número ?des? en el que se descompone el derivado paramétrico, como puntero constante ?far? a una cadena de caracteres. La propia función es responsable de asignar memoria al puntero. En caso de error, la función devuelve NULL. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

BOOL EXPORTA BdcRendimiento (HANDLE h, // identificador del concepto LONG des, // número del elemento de la descomposición double FAR *ren // rendimiento a obtener);

Propósito

Obtiene el rendimiento del elemento número ?des? en el que se descompone el derivado paramétrico.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

des: Número del elemento de la descomposición del concepto. Los elementos se numeran empezando por cero.

*ren: Puntero en el que devolver el rendimiento deseado. El rendimiento puede ser positivo, cero o negativo.

Valor devuelto

Devuelve ?0? si se ejecuta correctamente. En caso de error, el rendimiento se asigna a cero y la función devuelve ?-1?. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

BOOL EXPORTA BdcPrecio (HANDLE h, // identificador del concepto double FAR *pre // precio unitario a devolver);

Propósito

Obtiene el precio unitario en el caso de que el derivado paramétrico sea un simple. Es posible que un mismo concepto paramétrico tenga como derivados tanto simples como compuestos.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

*pre: Puntero en el que devolver el precio unitario. Dicho precio puede ser positivo, cero o negativo.

Valor devuelto

Devuelve ?0? si se ejecuta correctamente. En caso de error, el precio se asigna a cero y la función devuelve ?-1?. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

LPCSTR EXPORTA BdcCodigo (HANDLE h // identificador del concepto);

Propósito

Obtiene el CODIGO del concepto.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

Valor devuelto

Devuelve el CODIGO del concepto, como puntero constante ?far? a una cadena de caracteres. Si se ha calculado un derivado paramétrico (se ha llamado a BdcCalcula), este CODIGO será el del derivado paramétrico. En caso contario, será el CODIGO del concepto paramétrico. La propia función es responsable de asignar memoria al puntero. En caso de error, la función devuelve NULL. Para obtener más INFORMACION sobre el error producido, llame a la función

BdcError().

LPCSTR EXPORTA BdcResumen (HANDLE h // identificador del concepto);

Propósito

Obtiene el texto resumido del derivado paramétrico.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

Valor devuelto

Devuelve el texto resumido del derivado paramétrico, como puntero constante ?far? a una cadena de caracteres. La propia función es responsable de asignar memoria al puntero. Si no existe definido un texto resumido, la función devuelve la cadena vacía "". En caso de error, la función devuelve NULL. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

LPCSTR EXPORTA BdcTexto (HANDLE h // identificador del concepto);

Propósito

Obtiene el texto completo de descripción del derivado paramétrico.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

Valor devuelto

Devuelve el texto completo de descripción del derivado paramétrico, como puntero constante ?far? a una cadena de caracteres. La propia función es responsable de asignar memoria al puntero. Si no existe definido un texto completo de descripción, la función devuelve la cadena vacía "". En caso de error, la función devuelve NULL. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

LPCSTR EXPORTA BdcPliego (HANDLE h // identificador del concepto);

Propósito

Obtiene el texto del pliego del derivado paramétrico.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

Valor devuelto

Devuelve el texto del pliego del derivado paramétrico, como puntero constante ?far? a una cadena de caracteres. La propia función es responsable de asignar memoria al puntero. Si no existe definido un texto de pliego, la función devuelve la cadena vacía ""‥ En caso de error, la función devuelve NULL. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

LPCSTR EXPORTA BdcClaves (HANDLE h // identificador del concepto);

Propósito

Obtiene las claves de tesauro del derivado paramétrico.

Parámetros

h: Identificador (HANDLE) del concepto paramétrico, que debe ser obtenido en una llamada anterior a la función BdcLee().

Valor devuelto

Devuelve las claves de tesauro del derivado paramétrico, como puntero constante ?far? a una cadena de caracteres, con el mismo formato que el registro ?~A?, es decir, ?{CLAVE_TESAURO\}?. La propia función es responsable de asignar memoria al puntero. Si no existen definidas claves del tesauro, la función devuelve la cadena vacía "". En caso de error, la función devuelve NULL. Para obtener más INFORMACION sobre el error producido, llame a la función BdcError().

4. MENSAJES DE ERROR

Los CODIGOs de error se almacenan en un LONG (entero de 32 bits) de manera que cada error corresponde a un bit. De esta forma, es posible definir hasta 32 CODIGOs de error que pueden producirse de forma aislada o conjunta. Las

llamadas a las funciones BdcGloError() y BdcError() eliminan los CODIGOs de error porducidos anteriormente.

Por ejemplo, para saber si un determinado error se ha producido con el concepto ?Concepto?, se debe utilizar la sintaxis:

```
const char *Mensaje;
BOOL cod_err = BdcError ((HANDLE)Concepto, &Mensaje);
if (cod_err & BDCERR_BASE_DATOS) {
// Se ha producido el error ?BDCERR_BASE_DATOS?
...
}
```

4.1. CODIGOs de los mensajes de error

BDCERR_CORRECTO No hay error.

BDCERR_BASE_DATOS Existe un mensaje de error. Es el caso en el que la DEFINICION paramétrica se indica una combinación inválida y se devuelve un mensaje de error explicativo.

BDCERR_PARAMETRO Se pasó a BdcCalcula o BdcGloCalcula un parámetro inexistente.

BDCERR_OPCION Se pasó a BdcCalcula o BdcGloCalcula una opción inexistente.

BDCERR_MAX_OPCIONES Se definieron más de 62 opciones en un determinado parámetro.

BDCERR_NO_LEIDO Se intentó utilizar BdcCalcula() antes que BdcLee().

BDCERR_NO_CALCULADO Se intentó acceder a datos de un derivado paramétrico antes de utilizar BdcCalcula().

BDCERR_DESCOMPOSICION Se intentó acceder a un elemento de la descomposición inexistente.

BDCERR_SIN_CODIGO No existe CODIGO definido.

BDCERR_SIN_MEMORIA Memoria insuficiente.

BDCERR_CONCEPTO_NULO Se pasó un HANDLE nulo.

EJEMPLOS

1. BASE DE DATOS EJEMPLO: BASE.DLL

1.1. Archivos para la distribución de la Base de Datos.

Para distribuir la base datos que se construyera con este ejemplo, se deberían proporcionar los siguientes archivos:

base.dll DLL que contiene tanto las descripciones paramétricas y como el interfaz ESTANDAR con aplicaciones (API).

base.bc3 Archivo ASCII de la base de datos en formato FIEBDC-3/98. En el ejemplo, la base incluye en esta DLL el paramétrico global de la base, así como las descripciones paramétricas de los conceptos "ABPH.1$", "SBRG.1$" y "EADR.3$", por lo que al menos debe contar con los siguientes registros:

~P| | | BASE.DLL |

~P| ABPH.1$ | |

~P| SBRG.1$ | |

~P| EADR.3$ | |

1.2. Archivos necesarios para la CONSTRUCCION de la Base de Datos.

Este ejemplo está preparado para compilarse con Microsoft Visual C++ VERSION 2.2 o posterior como DLL de 32 bits. Sin apenas modificaciones, sería posible compilarlo como DLL de 16 bits o utilizar el compilador Borland C++.

Para contruir la DLL, son necesarios los siguientes archivos:

fiebdc.h Archivo que define el formato, con el contenido ya especificado.

base.h DEFINICION de variables y defines útiles para la defínción de las descripciones

paramétricas.

interfaz.cpp Implementación de las funciones del API.

aplicat.cpp Implementación de las funciones de la descripción paramétrica.

base.cpp Implementación de las descripciones paramétricas de la base de datos en formato C++. Es la única parte que escribirían los redactores de las bases de datos. En el ejemplo se incluyen dos descompuestos y un elemental. En el elemental se ha utilizado la posibilidad que posee el formato de que el CODIGO del derivado paramétrico no refleje directamente el valor de sus parámetros. Se ha utilizado una sintaxis similar a la descripción paramétrica del formato FIEBDC-3/95, para facilitar así el intercambio entre ambos formatos.

base.def DEFINICION de las funciones de exportación del API.

1.2.1. Archivo ?base.h?

```
#define BASE
#include "Fiebdc.h"

#define PAR par.parametro
#define PRE par.precio
#define DES par.descompuesto
#define RES par.resumen
#define TEX par.texto
#define PLI par.pliego
#define ROT par.lee_rotulo
#define ERR par.error
#define COD par.codigo
#define SIN par.sinonimo
#define CLA par.claves
#define COM par.comentario
#define INI(Op) par.inicializa(Op)

#define ROTA ROT(0)
#define ROTB ROT(1)
#define ROTC ROT(2)
#define ROTD ROT(3)

// Parámetros de cada concepto
#define A par.lee_opcion(0)
#define B par.lee_opcion(1)
#define C par.lee_opcion(2)
```

```
#define D par.lee_opcion(3)
// Parámetros del paremétrico Global
#define O Cglobal->lee_opcion(0)
#define P Cglobal->lee_opcion(1)
#define Q Cglobal->lee_opcion(2)
#define R Cglobal->lee_opcion(3)

// Valores de status
#define ST_VACIO 0
#define ST_FAMILIA 1
#define ST_DERIVADO 2
// Valores de operación
#define BORRA 0
#define LEE 1
#define CALCULA 2
#define BUSCA 3
#define VALIDA 4

// Valores BOOL de devolución
#define CORRECTO 0
#define ERRONEO -1

#define MAX_PARAM 12 // Máximo número de paránetros
#define MAX_TXT 30000 // Longitud máxima de los textos

class Cfiebdc;
typedef short (*PRECIO)(Cfiebdc &, short operacion);

extern Cfiebdc *Cglobal; // Paramétrico Global
extern PRECIO precios[]; // Lista de precios (conceptos) de la base
extern short Global (Cfiebdc &, short operacion); // Función del paramétrico Global

class Cfiebdc {
private:
long parnum; // Nº de parámetros
long opcnum[MAX_PARAM]; // Nº de opciones por parámetro
char *parrot[MAX_PARAM]; // Texto de los rótulos de cada parámetro
char **opcrot[MAX_PARAM]; // Texto de las opciones de cada parámetro
```

```
long opc[MAX_PARAM]; // Valores de cada parámetro

char *err; // Texto de error
double pre; // Precio unitario (precio simple)
long desnum; // Nº de descompuestos
char **descod; // CODIGOs de cada concpto de la descomposición
double *desren; // Rendimiento de cada concpto de la descomposición
char *res, *tex, *pli; // Texto resumido, Completo y Pliego
char *cla, *com; // Claves y comentario
char *codfam; // CODIGO del concepto paramétrico (familia)
char *subcod; // Parte variable del CODIGO automática
char *codder; // CODIGO completo del derivado
BOOL status; // Estado: ST_VACIO, ST_FAMILIA, ST_DERIVADO
long coderr; // CODIGO de error

long sinnum; // Nº de sinónimos
char **codsin[2]; // Textos [0] subCODIGOs automáticos [1] sinónimos

public:
PRECIO funcion; // Puntero a la función del precio
BOOL parametro (char *Rot, ...); // Crea un parámetro
BOOL precio (double Pre); // Fija el precio unitario
BOOL descompuesto (double Ren, char *cod, ...); // Crea un elemento de
// la descomposición
BOOL resumen (char *Res, ...); // Fija el texto resumido
BOOL texto (char *Tex, ...); // Fija el texto de descripción
BOOL pliego (char *Pli, ...); // Fija el texto del pliego
BOOL comentario (char *Com, ...); // Fija el texto del comentario
BOOL claves (char *Cla, ...); // Fija el texto de las claves
BOOL error (char *Err, ...); // Fija el texto de error
BOOL codigo (char *Cod, ...); // Fija el CODIGO (familia o derivado)
void estado (BOOL Status) {status = Status; };
void cod_error (BOOL CodErr) {coderr |= CodErr; };
BOOL subcodigo (void); // Calcula el subCODIGO
BOOL opciones (long *Opc); // Fija los valores de los parámetros
BOOL opciones_glo (long ParGNum, long *OpcG); // Fija los valores de
// los parámetros globales
BOOL sinonimo (char *Subcod, char *Codsin); // Fija el sinónimo se Subcod
```

```
long lee_num_par (void) { return parnum; };
long lee_num_des (void) { return desnum; };
long lee_num_opc (long id_par) { return opcnum[id_par];};
char *lee_rot_par (long id_par);
char *lee_rot_opc (long id_par, long id_opc);
char *lee_cod_des (long id_des);
double lee_ren_des (long id_des);
double lee_precio (void) { return pre; };
char *lee_codigo (void);
char *lee_resumen (void);
char *lee_texto (void);
char *lee_pliego (void);
char *lee_claves (void);
char *lee_comentario (void);
char *lee_error (void);
long lee_opcion (long Par); // Devuelve la opción fijada en el
// parámetro Par
char *lee_rotulo (long Par); // Devuelve el rótulo de la opción fijada en
// el parámetro Par
BOOL lee_opciones (char *Cod, long *Opc); // Devuelve las opciones de
// un CODIGO
BOOL lee_estado (void) { return status; };
BOOL lee_cod_error (void) { return coderr; };
int inicializa (short operacion);
Cfiebdc ();
~Cfiebdc ();
};
```

1.2.2. Archivo ?inferfaz.cpp?

```
#define STRICT
#include <windows.h>
#include <string.h>
#include "Base.h"

Cfiebdc *Cglobal;

////////////////////////////////////////////////////////////////////////////
// FUNCIONES GENERALES Y DE INICIALIZACIÓN
```

```
BOOL APIENTRY
DllMain ( // Específico de Microsoft. Borland utiliza ?DllEntryPoint?
HANDLE hModule,
DWORD tipo_llamada,
LPVOID lpReserved )
{
switch (tipo_llamada) {
case DLL_PROCESS_ATTACH:
case DLL_THREAD_ATTACH:
// Inicialización
Cglobal = new Cfiebdc;
Global (*Cglobal, LEE);
Cglobal->funcion = &Global;
break;
case DLL_THREAD_DETACH:
case DLL_PROCESS_DETACH:
// Liberación
Cglobal->inicializa (BORRA);
Cglobal->Cfiebdc::~Cfiebdc();
}
return TRUE;
}

LONG EXPORTA
BdcCodificacion (VOID)
{
return 1; // La codificación es indipendiente de los valores de los parámetros
}

////////////////////////////////////////////////////////////////////////////
// FUNCIONES REFERENTES AL PARAMÉTRICO GLOBAL

// Obtención de sus parámetros
LONG EXPORTA
BdcGloParNumero (VOID)
{
return BdcParNumero (Cglobal);
}
```

```
LONG EXPORTA
BdcGloOpcNumero (
LONG par )
{
return BdcOpcNumero (Cglobal, par);
}

LPCSTR EXPORTA
BdcGloParRotulo (
LONG par )
{
return BdcParRotulo (Cglobal, par);
}

LPCSTR EXPORTA
BdcGloOpcRotulo (
LONG par,
LONG opc )
{
return BdcOpcRotulo (Cglobal, par, opc);
}

BOOL EXPORTA
BdcGloCalcula (
LPLONG opcl)
{
return BdcCalcula (Cglobal, opcl);
}

BOOL EXPORTA
BdcGloError (
LPCSTR *err)
{
return BdcError (Cglobal, err);
}
////////////////////////////////////////////////////////////////////////////
// FUNCIONES REFERENTES AL RESTO DE PARAMÉTRICOS

// Lectura de un concepto paramétrico
```

```
HANDLE EXPORTA
BdcLee (
LPCSTR cod) {
int i;

Cfiebdc *Cpar = new Cfiebdc();
for (i=0;;i++) {
Cpar->inicializa(BORRA);
if (!((precios [i])(*Cpar, BUSCA)) )
break;
if (strcmp (Cpar->lee_codigo(), cod) == 0) {
(precios [i])(*Cpar, LEE);
Cpar->funcion = precios[i];
return (HANDLE)Cpar;
}
}
BdcCierra ( (HANDLE)Cpar );
return (HANDLE) 0; // El precio no existe
}

// Lectura de un concepto paramétrico a partir del CODIGO del derivado paramétrico
HANDLE EXPORTA
BdcDecodifica (
LPCSTR cod,
LPLONG opcl )
{
int i,j;
char cod_fam[8];

if (strlen(cod)<7) return (HANDLE)0;
strncpy (cod_fam, cod, 6);
cod_fam[6]='$';
cod_fam[7]='\0';
for (i=0; i<7; i++)
if (cod_fam[i]<=' ') cod_fam[i]='_';
Cfiebdc *Cpar = new Cfiebdc();
for (i=0;;i++) {
Cpar->inicializa(BORRA);
if (!((precios [i])(*Cpar, BUSCA)) )
```

```
break;
if (strcmp (Cpar->lee_codigo(), cod_fam) == 0) {

//Se encontró el paramétrico
(precios [i])(*Cpar, LEE);
Cpar->funcion = precios[i];

j=Cpar->lee_num_par();

if (!Cpar->lee_opciones((char *)cod+6, opcl)) {
break;
}
return (HANDLE)Cpar;
}
}
BdcCierra ( (HANDLE)Cpar );
return (HANDLE) 0; // El precio no existe
}
// Obtención de sus parámetros
LONG EXPORTA
BdcParNumero (
HANDLE h )
{
Cfiebdc *Cpar;

if (!h) return ERRONEO;
else Cpar = (Cfiebdc *)h;
if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return ERRONEO;
} else
return Cpar->lee_num_par();
}

LONG EXPORTA
BdcOpcNumero (
HANDLE h,
LONG par)
{
```

```
Cfiebdc *Cpar;
if (!h) return ERRONEO;
else Cpar = (Cfiebdc *)h;
if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return ERRONEO;
} else
return Cpar->lee_num_opc (par);
}

LPCSTR EXPORTA
BdcParRotulo (
HANDLE h,
LONG par)
{
Cfiebdc *Cpar;
if (!h) return NULL;
else Cpar = (Cfiebdc *)h;
if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return NULL;
} else
return (LPCSTR)Cpar->lee_rot_par(par);
}

LPCSTR EXPORTA
BdcOpcRotulo (
HANDLE h,
LONG par,
LONG opc)
{
Cfiebdc *Cpar;
if (!h) return NULL;
else Cpar = (Cfiebdc *)h;
if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return NULL;
} else
return (LPCSTR)Cpar->lee_rot_opc (par, opc);
```

```
}

// Asignación de opciones de los parámetros y comprobación de su validez
BOOL EXPORTA
BdcValida (
HANDLE h,
LPLONG opcl )
{
Cfiebdc *Cpar;
if (!h) return ERRONEO;
else Cpar = (Cfiebdc *)h;
if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return ERRONEO;
}
if (!Cpar->opciones (opcl)) return FALSE;
return ((Cpar->funcion) (*Cpar, VALIDA)) ? CORRECTO : ERRONEO;
}

// Asignación de opciones de los parámetros y cálculo del derivado
BOOL EXPORTA
BdcCalcula (
HANDLE h,
LPLONG opcl )
{
Cfiebdc *Cpar;
if (!h) return ERRONEO;
else Cpar = (Cfiebdc *)h;
if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return ERRONEO;
}
if (!Cpar->opciones (opcl)) return FALSE;
return ((Cpar->funcion) (*Cpar, CALCULA)) ? CORRECTO : ERRONEO;
}

// Liberación de memoria
BOOL EXPORTA
BdcCierra (
```

```
HANDLE h)
{
Cfiebdc *Cpar;
if (!h) return ERRONEO;
else Cpar = (Cfiebdc *)h;
Cpar->inicializa (BORRA);
Cpar->Cfiebdc::~Cfiebdc();
return CORRECTO;
}

// Obtención del derivado paramétrico
LONG EXPORTA
BdcDesNumero (
HANDLE h )
{
Cfiebdc *Cpar;
if (!h) return ERRONEO;
else Cpar = (Cfiebdc *)h;

if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return ERRONEO;
}
if (Cpar->lee_estado() != ST_DERIVADO) {
Cpar->cod_error (BDCERR_NO_CALCULADO);
return ERRONEO;
}
return Cpar->lee_num_des();
}

LPCSTR EXPORTA
BdcDesCodigo (
HANDLE h,
LONG des )
{
Cfiebdc *Cpar;

if (!h) return NULL;
else Cpar = (Cfiebdc *)h;
```

```
if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return NULL;
}
if (Cpar->lee_estado() != ST_DERIVADO) {
Cpar->cod_error (BDCERR_NO_CALCULADO);
return NULL;
}
return (LPCSTR)Cpar->lee_cod_des (des);
}

BOOL EXPORTA
BdcRendimiento (
HANDLE h,
LONG des,
double FAR *ren)
{
Cfiebdc *Cpar;

*ren = 0.0;
if (!h) return ERRONEO;
else Cpar = (Cfiebdc *)h;

if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return ERRONEO;
}
if (Cpar->lee_estado() != ST_DERIVADO) {
Cpar->cod_error (BDCERR_NO_CALCULADO);
return ERRONEO;
}
*ren = Cpar->lee_ren_des(des);
return CORRECTO;
}

BOOL EXPORTA
BdcPrecio (
HANDLE h,
```

```
double FAR *pre)
{
Cfiebdc *Cpar;
*pre=0.0;

if (!h) return ERRONEO;
else Cpar = (Cfiebdc *)h;

if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return ERRONEO;
}
if (Cpar->lee_estado() != ST_DERIVADO) {
Cpar->cod_error (BDCERR_NO_CALCULADO);
return ERRONEO;
}
*pre = Cpar->lee_precio();
return CORRECTO;
}

LPCSTR EXPORTA
BdcCodigo (
HANDLE h)
{
Cfiebdc *Cpar;

if (!h) return NULL;
else Cpar = (Cfiebdc *)h;

if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return NULL;
}
return (LPCSTR)Cpar->lee_codigo();
}

LPCSTR EXPORTA
BdcResumen (
HANDLE h)
```

```
{
Cfiebdc *Cpar;

if (!h) return NULL;
else Cpar = (Cfiebdc *)h;

if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return NULL;
}
if (Cpar->lee_estado() != ST_DERIVADO) {
Cpar->cod_error (BDCERR_NO_CALCULADO);
return NULL;
}
return (LPCSTR)Cpar->lee_resumen();
}

LPCSTR EXPORTA
BdcTexto (
HANDLE h)
{
Cfiebdc *Cpar;

if (!h) return NULL;
else Cpar = (Cfiebdc *)h;

if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return NULL;
}
if (Cpar->lee_estado() != ST_DERIVADO) {
Cpar->cod_error (BDCERR_NO_CALCULADO);
return NULL;
}
return (LPCSTR)Cpar->lee_texto();
}

LPCSTR EXPORTA
BdcPliego (
```

```
HANDLE h)
{
Cfiebdc *Cpar;

if (!h) return NULL;
else Cpar = (Cfiebdc *)h;

if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return NULL;
}
if (Cpar->lee_estado() != ST_DERIVADO) {
Cpar->cod_error (BDCERR_NO_CALCULADO);
return NULL;
}
return (LPCSTR)Cpar->lee_pliego();
}
LPCSTR EXPORTA
BdcComentario (
HANDLE h)
{
Cfiebdc *Cpar;

if (!h) return NULL;
else Cpar = (Cfiebdc *)h;

if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return NULL;
}
return (LPCSTR)Cpar->lee_comentario();
}

LPCSTR EXPORTA
BdcClaves (
HANDLE h)
{
Cfiebdc *Cpar;
```

```
if (!h) return NULL;
else Cpar = (Cfiebdc *)h;

if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
return NULL;
}
if (Cpar->lee_estado() != ST_DERIVADO) {
Cpar->cod_error (BDCERR_NO_CALCULADO);
return NULL;
}
return (LPCSTR)Cpar->lee_claves();
}

BOOL EXPORTA
BdcError (
HANDLE h,
LPCSTR *err)
{
Cfiebdc *Cpar;
BOOL ret;

if (!h) { *err=""; return BDCERR_CONCEPTO_NULO; }
else Cpar = (Cfiebdc *)h;

if (Cpar->lee_estado() == ST_VACIO) {
Cpar->cod_error (BDCERR_NO_LEIDO);
*err = "";
return Cpar->lee_cod_error();
}
*err = (LPCSTR)Cpar->lee_error();
ret = Cpar->lee_cod_error();
Cpar->cod_error (BDCERR_CORRECTO);
return ret;
}
```

1.2.3. Archivo ?aplicat.cpp?

```
#define STRICT
#include <windows.h>
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>

#include "Base.h"

Cfiebdc::Cfiebdc (void) {
int i,j;

estado (ST_VACIO);
coderr =BDCERR_CORRECTO;
// parámetros y opciones por parámetro
for (i=0; i<MAX_PARAM; i++) {
parrot[i] = (char *)0;
opcrot[i] = (char **)0;
}

parnum=0;
pre=0.;

// sinónimos
for (j=0; j<2; j++) {
codsin[j]= (char **)0;
}
sinnum=0;

// descomposición
desren = (double *)0;
descod = (char **)0;
desnum=0;
// textos
err = (char *)0;
res = (char *)0;
tex = (char *)0;
pli = (char *)0;
com = (char *)0;
```

```
cla = (char *)0;
codfam = (char *)0;
subcod = (char *)0;
codder = (char *)0;
}

Cfiebdc::~Cfiebdc (void) {
estado (ST_VACIO);
}

Cfiebdc::inicializa (
short operacion )
{
int i,j;

coderr = BDCERR_CORRECTO;

if (operacion == BORRA) {
estado (ST_VACIO);
// parámetros y opciones por parámetro
for (i=0; i<parnum; i++) {
if (parrot[i]) {
delete[] parrot[i];
parrot[i] = (char *)0;
}
for (j=0; j< opcnum[i]; j++) {
if (opcrot[i][j]) {
delete[] opcrot[i][j];
opcrot[i][j]= (char *)0;
}
}
if (opcrot[i]) {
delete[] opcrot[i];
opcrot[i]= (char **)0;
}
}

parnum=0;
pre=0.;
```

```
// sinónimos
for (j=0; j<2; j++) {
for (i=0; i<sinnum; i++) {
if (codsin[j][i]) {
delete[] codsin[j][i];
codsin[j][i]= (char *)0;
}
}
if (codsin[j]) {
delete[] codsin[j];
codsin[j]= (char **)0;
}
}
sinnum=0;

// descomposición
if (desren) {
delete[] desren;
desren = (double *)0;
}
for (i=0; i<desnum; i++) {
if (descod[i]) {
delete[] descod[i];
descod[i] = (char *)0;
}
}
if (descod) {
delete[] descod;
descod = (char **)0;
}
desnum=0;
// textos
if (err) { delete[] err; err = (char *)0; }
if (res) { delete[] res; res = (char *)0; }
if (tex) { delete[] tex; tex = (char *)0; }
if (pli) { delete[] pli; pli = (char *)0; }
if (cla) { delete[] cla; cla = (char *)0; }
if (com) { delete[] com; com = (char *)0; }
```

```
if (codfam) { delete[] codfam; codfam = (char *)0; }
if (subcod) { delete[] subcod; subcod = (char *)0; }
if (codder) { delete[] codder; codder = (char *)0; }
}

else if (operacion == LEE) {
estado (ST_FAMILIA);
}

else if (operacion==CALCULA) {
estado (ST_DERIVADO);
pre=0.;

// descomposición
if (desren) {
delete[] desren;
desren = (double *)0;
}
for (i=0; i<desnum; i++) {
if (descod[i]) {
delete[] descod[i];
descod[i] = (char *)0;
}
}
if (descod) {
delete[] descod;
descod = (char **)0;
}
desnum=0;
if (err) { delete[] err; err = (char *)0; }
if (res) { delete[] res; res = (char *)0; }
if (tex) { delete[] tex; tex = (char *)0; }
if (pli) { delete[] pli; pli = (char *)0; }
if (cla) { delete[] cla; cla = (char *)0; }
if (codfam) { delete[] codfam; codfam = (char *)0; }
if (subcod) { delete[] subcod; subcod = (char *)0; }
if (codder) { delete[] codder; codder = (char *)0; }
}
return TRUE;
```

```
}

BOOL
Cfiebdc::parametro (
char *rot, ... )
{
int i, len;
long Opcnum=0;
char *s;
va_list marker;

if (parnum>=MAX_PARAM) return FALSE;

len = strlen(rot);
parrot[parnum] = new char[len+1];
strcpy (parrot[parnum], rot);

va_start( marker, rot );
do {
s = va_arg( marker, char *);
len = strlen (s);
if (len==0) break;
Opcnum++;
} while ( s[0] != '\0' );

va_start( marker, rot );
opcrot[parnum] = new char* [Opcnum];
for (i=0; i<Opcnum; i++) {
s = va_arg( marker, char *);
len = strlen (s);
opcrot[parnum][i] = new char[len+1];
strcpy (opcrot[parnum][i], s);
}

va_end( marker );

opcnum[parnum]=Opcnum;
parnum++;
```

```
return TRUE;
}
BOOL
Cfiebdc::precio (
double Pre )
{
pre=Pre;
return TRUE;
}

BOOL
Cfiebdc::descompuesto (
double Ren,
char *Cod,... )
{
int i, len;
char s[255], **Descod;
va_list marker;
double *Desren;

if (fabs(Ren)<0.0005) return FALSE;

Desren = new double [desnum+1];
memcpy (Desren, desren, desnum*sizeof(double));
delete[] desren;
desren = Desren;
desren [desnum] = Ren;

Descod = new char *[desnum+1];
for (i=0; i<desnum; i++)
Descod[i] = descod[i];
delete[] descod;
descod = Descod;
va_start( marker, Cod );

vsprintf (s,Cod,marker);

len = strlen (s);
descod[desnum] = new char[len+1];
```

```
strcpy (descod[desnum], s);

va_end( marker );
desnum++;
return TRUE;
}

BOOL
Cfiebdc::resumen (
char *Res, ... )
{
char *s = new char [MAX_TXT];
int len;
va_list marker;

va_start( marker, Res );

vsprintf (s,Res,marker);

len = strlen (s);
res = new char[len+1];
strcpy (res, s);
delete[] (s);

va_end( marker );

return TRUE;
}

BOOL
Cfiebdc::texto (
char *Tex, ...)
{
char *s = new char [MAX_TXT];
int len;
va_list marker;

va_start( marker, Tex );
```

```
vsprintf (s,Tex,marker);

len = strlen (s);
tex = new char[len+1];
strcpy (tex, s);
delete[] (s);

va_end( marker );

return TRUE;
}

BOOL
Cfiebdc::pliego (
char *Pli, ...)
{
char *s = new char [MAX_TXT];
int len;
va_list marker;

va_start( marker, Pli );

vsprintf (s,Pli,marker);

len = strlen (s);
pli = new char[len+1];
strcpy (pli, s);
delete[] (s);

va_end( marker );

return TRUE;
}

BOOL
Cfiebdc::claves (
char *Cla, ...)
{
char *s = new char [MAX_TXT];
```

```
int len;
va_list marker;

va_start( marker, Cla );

vsprintf (s,Cla,marker);

len = strlen (s);
pli = new char[len+1];
strcpy (cla, s);
delete[] (s);

va_end( marker );

return TRUE;
}

BOOL
Cfiebdc::comentario (
char *Com, ...)
{
char *s = new char [MAX_TXT];
int len;
va_list marker;

va_start( marker, Com );

vsprintf (s,Com,marker);

len = strlen (s);
pli = new char[len+1];
strcpy (com, s);
delete[] (s);

va_end( marker );

return TRUE;
}
```

```
BOOL
Cfiebdc::error (
char *Err, ...)
{
char *s = new char [MAX_TXT];
int len;
va_list marker;

va_start( marker, Err );

vsprintf (s,Err,marker);

len = strlen (s);
err = new char[len+1];
strcpy (err, s);
delete[] (s);

va_end( marker );

cod_error (BDCERR_BASE_DATOS);
return FALSE;
}

long
Cfiebdc::lee_opcion (
long Par)
{
if (Par>=parnum) {
cod_error (BDCERR_PARAMETRO);
return -1;
}
return opc[Par];
}

char
*Cfiebdc::lee_rotulo (
long Par)
{
if (Par>=parnum || opc[Par]>=opcnum[Par]) {
```

```
cod_error (BDCERR_PARAMETRO);
return NULL;
}
return opcrot[Par][opc[Par]];
}

BOOL
Cfiebdc::codigo (
char *Cod, ...)
{
char *s = new char [MAX_TXT];
int i,len;
va_list marker;

va_start( marker, Cod );

vsprintf (s,Cod,marker);

len = strlen (s);
for (i=0; i<len; i++)
if (s[i] == '$') break;

if (codfam) delete[] codfam;
if (i<len) {
codfam = new char[len+1];
strcpy (codfam, s);
} else {
codfam = new char[len+2];
sprintf (codfam, "%s%c", s, '$');
}
delete[] s;

va_end( marker );

if (status==ST_DERIVADO)
return subcodigo();
return TRUE;
}
```

```
BOOL
Cfiebdc::opciones (
long *Opc )
{
int i;
BOOL ret=TRUE;

for (i=0; i<parnum; i++) {
if (Opc[i]<0 || Opc[i]>=opcnum[i]) {
cod_error (BDCERR_OPCION);
opc[i] = 0;
ret=FALSE;
} else {
opc[i] = Opc[i];
}
}
return ret;
}

BOOL
Cfiebdc::sinonimo (
char *Subcod,
char *Codsin )
{
char **Sin[2];
int i,j,len1,len2;

len1 = strlen (Subcod);
len2 = strlen (Codsin);

if (!len1 || !len2) return FALSE;

for (i=0; i<2; i++) {
Sin[i] = new char *[sinnum+1];
for (j=0; j<sinnum; j++) {
Sin[i][j] = codsin[i][j];
}
if (codsin[i]) delete[] codsin[i];
}
```

```
Sin[0][sinnum] = new char [len1+1];
Sin[1][sinnum] = new char [len2+1];
strcpy (Sin[0][sinnum], Subcod);
strcpy (Sin[1][sinnum], Codsin);

codsin[0]=Sin[0];
codsin[1]=Sin[1];

sinnum++;

return TRUE;
}

char
*Cfiebdc::lee_rot_par (
long id_par)
{
if (id_par >= parnum || !parrot[id_par]) {
cod_error (BDCERR_PARAMETRO);
return "";
}
else
return parrot[id_par];
}

char
*Cfiebdc::lee_rot_opc
(long id_par,
long id_opc) {
if (id_par >= parnum) {
cod_error (BDCERR_PARAMETRO);
return NULL;
} else if (id_opc >= opcnum[id_par] || !opcrot[id_par] || !opcrot[id_par][id_opc]) {
cod_error (BDCERR_OPCION);
return NULL;
} else
return opcrot[id_par][id_opc];
}
```

```
char
*Cfiebdc::lee_cod_des (
long id_des )
{
if (id_des >= desnum || !descod[id_des]) {
cod_error (BDCERR_DESCOMPOSICION);
return NULL;
} else
return descod[id_des];
}

double
Cfiebdc::lee_ren_des (
long id_des )
{
if (id_des >= desnum) {
cod_error (BDCERR_DESCOMPOSICION);
return -1.0;
} else
return desren[id_des];
}

char
*Cfiebdc::lee_codigo (void) {
if (codder)
return codder;
else if (codfam)
return codfam;
else {
cod_error (BDCERR_SIN_CODIGO);
return NULL;
}
}

char
*Cfiebdc::lee_resumen (void) {
if (res) return res;
else return "";
}
```

```
char
*Cfiebdc::lee_texto (void) {
if (tex) return tex;
else return "";
}
char
*Cfiebdc::lee_pliego (void) {
if (pli) return pli;
else return "";
}
char
*Cfiebdc::lee_claves (void) {
if (pli) return cla;
else return "";
}
char
*Cfiebdc::lee_comentario (void) {
if (pli) return com;
else return "";
}

char
*Cfiebdc::lee_error (void) {
if (err) return err;
else return "";
}

BOOL
Cfiebdc::subcodigo (void) {
int i, az='z'-'a', AZ='Z'-'A';
BOOL ret=TRUE;
char *s = new char [MAX_TXT];
// Obtención del subCODIGO automático
subcod = new char [parnum+1];
for (i=0; i<parnum; i++) {
if (opc[i]>=0 && opc[i]<=0+az)
subcod[i]=opc[i]+'a';
else if (opc[i]>0+az && opc[i]<=0+az+AZ)
```

```
subcod[i]=opc[i]-az+’A’;
else if (opc[i]>0+az+AZ && opc[i]<=9+az+AZ)
subcod[i]=opc[i]-az-AZ;
else {
subcod[i]=’_’;
cod_error (BDCERR_MAX_OPCIONES);
ret = FALSE;
}
}
subcod[i]=’\0’;
// Comprobar lista de sinónimos
for (i=0; i<sinnum; i++) {
if (strcmp(subcod,codsin[0][i]) == 0) {
delete [] subcod;
subcod = new char [strlen (codsin[1][i])+1];
strcpy (subcod, codsin[1][i]);
}
}
// Rellenar codfam y codder
strcpy (s, codfam);
for (i=0; i<(int)strlen(codfam); i++)
if (codfam[i]==’$’) break;
strcpy (s+i, subcod);

codder = new char [strlen(s)+1];
strcpy (codder, s);
return ret;
}

BOOL
Cfiebdc::lee_opciones (
char *Cod, // parte variable del CODIGO
long *Opc ) // opciones a devolver
{
int i, len, az=’z’-’a’, AZ=’Z’-’A’;
char *s;
BOOL ret=TRUE;

for (i=0; i<sinnum; i++) {
```

```
if (strcmp(Cod, codsin[1][i])==0) {
len = strlen (codsin[0][i]);
s = codsin[0][i];
break;
}
}
if (i==sinnum) {
s = Cod;
len = strlen (s);
}

for (i=0; i<len; i++) {
if (s[i] >= ’a’ && s[i] <= ’z’)
Opc[i] = s[i]-’a’;
else if (s[i] >= ’A’ && s[i] <= ’Z’)
Opc[i] = s[i]-’A’ + az;
else if (s[i] >= ’0’ && s[i] <= ’9’)
Opc[i] = s[i]-’0’ + az + AZ;
else {
// excedida capacidad
Opc[i] = az + AZ + 11;
cod_error (BDCERR_MAX_OPCIONES);
ret = FALSE;
}
}
return ret;
}
```

1.2.4. Archivo ?base.cpp?

```
#define STRICT
#include <windows.h>
#include <stdio.h>
#include <stdarg.h>
#include <math.h>

#include "Base.h"
```

```
////////////////////////////////////////////////////////////////////////////
// Concepto Global
//

short Global (Cfiebdc &par, short operacion) {
INI (operacion);

switch (operacion) {
case LEE:
PAR ("ÁMBITO","España","Madrid","Barcelona","Valencia","Bilbao","Sevilla","");
return TRUE;
}
return TRUE;
}

////////////////////////////////////////////////////////////////////////////
// Conceptos
//

short fABPH_1 (Cfiebdc &par, short operacion) {

INI (operacion);
COD ("ABPH.1$");

switch (operacion) {

case LEE:
PAR ("RESISTENCIA", "H-50", "H-100", "H-125", "H-150", "H-175", "");
PAR ("CONSISTENCIA", "plástica", "blanda", "");
PAR ("Tmax", "18", "38", "78", "");
return TRUE;

case CALCULA:
case VALIDA:
// plástica H50 H100 H125 H150 H175
double T[2][3][5] ={ .180, .255, .290, .330, .365, //18
.160, .225, .260, .290, .325, //38
.140, .200, .225, .255, .285, //78
```

```
// blanda
.210, .290, .330, .375, .000,
.185, .260, .300, .335, .375,
.165, .235, .265, .300, .335 };

if (!T[B][C][A]) return ERR ("Combinación no permitida");
if (operacion == VALIDA) return TRUE;

double U[2][3]= { .180, .160, .140,
.205, .185, .165 };
double V[2][3][5]= { .695, .675, .665, .650, .640,
.720, .700, .690, .680, .670,
.740, .725, .720, .710, .700,
.665, .640, .630, .615, .0,
.690, .670, .665, .645, .635,
.715, .695, .685, .775, .665 };
DES (T[B][C][A], "SBAC.5ccaa"); // Cemento
DES (U[B][C], "SBAA.1a"); // Agua
DES (V[B][C][A], "SBRA.5ab"); // Arena
DES (V[B][C][A]*2, "SBRG.1%c", 'a'+C); // Grava
DES (1, "MAMA19a"); // Hormigonera
DES (2, "MOOC13a"); // Peón ordinario
RES ("%s C/%s Tmax=%s mm", ROTA, ROTB, ROTC);
int G[5]= { 50, 100, 125, 150, 175 };
TEX ( "Hormigón %s, resistencia característica %d Kg/cm2, "
"consistencia %s, cemento tipo II-Z/35-A, arena silícea "
"(0/6) y grava silícea rodada, tamaño máximo %s mm, "
"confeccionado en obra con hormigonera de 300 l de capacidad. "
"Según RC-93 y EH-91.", ROTA, G[A], ROTB, ROTC );

return TRUE;
}

return TRUE;
}

short fSBRG_1 (Cfiebdc &par, short operacion) {
INI (operacion);
COD ("SBRG.1$");
```

```
switch (operacion) {

case LEE:
PAR ("Tmax", "18", "38", "78", "");
// Sinónimos
SIN ("a", "_18");
SIN ("b", "_38");
SIN ("c", "_78");
return TRUE;

case CALCULA:
// ÁMBITO (Primer parámetro global. Parámetro 'O'
// España Madrid Barcelona Valencia Bilbao Sevilla
double I [] = { 123.5, 135.1, 130.0, 129.8, 127.8, 125.9 };

PRE (I[O]+0.25*A);
RES ("Grava de Tmáx. %s", ROTA);

return TRUE;
}

return TRUE;
}

short fEADR_3 (Cfiebdc &par, short operacion) {
INI (operacion);
COD ("EADR.3$");
switch (operacion) {

case LEE:
PAR ("MATERIAL","yeso","cemento","cal","cal y cemento","");
PAR ("PARAMENTOS","verticales","horizontales","");
break;
case CALCULA:
char *cF;
// $A yeso cemento cal cal y cemento $B
double
pdT[2][4]= {{ 0.350, 0.500, 0.400, 0.450}, // vertical
```

```
{ 0.550, 0.800, 0.600, 0.650}}; // horizontal

DES (pdT[B][A], "MOOA.1d"); // peón ordinario
DES (0.02, "%%"); // maux

if (A==0) cF = "guarnecido";
else cF = "enfoscado";
char *pcG[4]= {"yeso", "cto", "cal", "cal y cto"};
char *pcH[2]= {"vert", "hrz"};

RES ("Picado %s param %s",pcG[A],pcH[B]);
TEX ("Picado de %s de %s en paramentos %s, retirada de escombros y carga, "
"sin transporte a vertedero.", cF, ROTA, ROTB);
}
return TRUE;
}
short fFIN (Cfiebdc &par, short operacion) {
return FALSE;
}

////////////////////////////////////////////////////////////////////////////
// Lista de Conceptos
//

PRECIO precios[] = {fABPH_1, fSBRG_1, fEADR_3, fFIN};
```

1.2.5. Archivo ?base.def?

```
LIBRARY BASE
DESCRIPTION ’Formato Fiebdc-3/98. Ejemplo de Definiciones paramétricas’
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD SINGLE

HEAPSIZE 1024

EXPORTS
BdcCodificacion
BdcGloParNumero
BdcGloOpcNumero
```

BdcGloParRotulo

BdcGloOpcRotulo

BdcGloCalcula

BdcGloError

BdcLee

BdcDecodifica

BdcParNumero

BdcOpcNumero

BdcParRotulo

BdcOpcRotulo

BdcComentario

BdcValida

BdcCalcula

BdcCierra

BdcDesNumero

BdcDesCodigo

BdcRendimiento

BdcPrecio

BdcCodigo

BdcResumen

BdcTexto

BdcPliego

BdcClaves

BdcError

2. APLICACION EJEMPLO: PROGRAMA.EXE

Esta sencilla aplicación, lee los conceptos que se incluyen en el propio fuente de la base BASE.DLL (en una aplicación real, los conceptos se definen en los registros ~P de base.bc3), y escribe en el archivo 'SALIDA.TXT' los rótulos de todos sus parámetros así como todos los datos de todas sus combinaciones paramétricas.

2.1. Archivos necesarios para la CONSTRUCCION del programa ejemplo.

Este ejemplo está preparado para compilarse con Microsoft Visual C++ VERSION 2.2 o posterior como aplicación Windows 95 de 32 bits en modo consola. Sin apenas modificaciones, sería posible compilarlo como aplicación de 16 bits o utilizar el compilador Borland C++.

Para construir la aplicación, son necesarios los siguientes archivos:

fiebdc.h Archivo que define el formato.

programa.h Declaración de variables y funciones.

program0.c Funciones auxiliares de apertura y cierre de la base de datos y tratamiento de los mensajes de error.

programa.c Programa ejemplo.

2.1.1. Archivo ?programa.h?

```
#include "Fiebdc.h"

#define CORRECTO 0
#define ERRONEO -1

BDCGLOPARNUMERO pGloParNumero;
BDCGLOOPCNUMERO pGloOpcNumero;
BDCGLOPARROTULO pGloParRotulo;
BDCGLOOPCROTULO pGloOpcRotulo;
BDCGLOCALCULA pGloCalcula;
BDCGLOERROR pGloError;

BDCLEE pLee;
BDCDECODIFICA pDecodifica;
BDCPARNUMERO pParNumero;
BDCOPCNUMERO pOpcNumero;
BDCPARROTULO pParRotulo;
BDCOPCROTULO pOpcRotulo;
BDCVALIDA pValida;
BDCCALCULA pCalcula;
BDCCIERRA pCierra;
BDCDESNUMERO pDesNumero;
BDCDESCODIGO pDesCodigo;
BDCRENDIMIENTO pRendimiento;
BDCPRECIO pPrecio;
BDCCODIGO pCodigo;
BDCRESUMEN pResumen;
```

```
BDCTEXTO pTexto;
BDCPLIEGO pPliego;
BDCCOMENTARIO pComentario;
BDCCLAVES pClaves;
BDCERROR pError;

BOOL AbreDll (char *nDll, HINSTANCE *hDll, char **Err);
BOOL CierraDll (HMODULE hDll, char **Err);
BOOL ChequeaError (HANDLE Precio, FILE *f, BOOL EsGlobal);
```

2.1.2. Archivo ?program0.c?

```
#define STRICT
#include <windows.h>
#include <stdio.h>
#include "programa.h"

BOOL
AbreDll (
char *nDll,
HINSTANCE *hDll,
char **Err )
{
*hDll = LoadLibrary (nDll);
if ( ! (*hDll) ) {
*Err = "IMPOSIBLE ABRIR LA LIBRERIA DE LA BASE";
return FALSE;
}
if( (pGloParNumero = (BDCGLOPARNUMERO) GetProcAddress (*hDll, "BdcGloParNumero")) &&
(pGloOpcNumero = (BDCGLOOPCNUMERO) GetProcAddress (*hDll, "BdcGloOpcNumero")) &&
(pGloParRotulo = (BDCGLOPARROTULO) GetProcAddress (*hDll, "BdcGloParRotulo")) &&
(pGloOpcRotulo = (BDCGLOOPCROTULO) GetProcAddress (*hDll, "BdcGloOpcRotulo")) &&
(pGloCalcula = (BDCGLOCALCULA) GetProcAddress (*hDll, "BdcGloCalcula")) &&
(pGloError = (BDCGLOERROR) GetProcAddress (*hDll, "BdcGloError")) &&
(pLee = (BDCLEE) GetProcAddress (*hDll, "BdcLee")) &&
(pDecodifica = (BDCDECODIFICA) GetProcAddress (*hDll, "BdcDecodifica")) &&
(pParNumero = (BDCPARNUMERO) GetProcAddress (*hDll, "BdcParNumero")) &&
(pOpcNumero = (BDCOPCNUMERO) GetProcAddress (*hDll, "BdcOpcNumero")) &&
```

```
(pParRotulo = (BDCPARROTULO) GetProcAddress (*hDll, "BdcParRotulo")) &&
(pOpcRotulo = (BDCOPCROTULO) GetProcAddress (*hDll, "BdcOpcRotulo")) &&
(pValida = (BDCVALIDA) GetProcAddress (*hDll, "BdcValida")) &&
(pCalcula = (BDCCALCULA) GetProcAddress (*hDll, "BdcCalcula")) &&
(pCierra = (BDCCIERRA) GetProcAddress (*hDll, "BdcCierra")) &&
(pDesNumero = (BDCDESNUMERO) GetProcAddress (*hDll, "BdcDesNumero")) &&
(pDesCodigo = (BDCDESCODIGO) GetProcAddress (*hDll, "BdcDesCodigo")) &&
(pRendimiento = (BDCRENDIMIENTO) GetProcAddress (*hDll, "BdcRendimiento")) &&
(pPrecio = (BDCPRECIO) GetProcAddress (*hDll, "BdcPrecio")) &&
(pCodigo = (BDCCODIGO) GetProcAddress (*hDll, "BdcCodigo")) &&
(pResumen = (BDCRESUMEN) GetProcAddress (*hDll, "BdcResumen")) &&
(pTexto = (BDCTEXTO) GetProcAddress (*hDll, "BdcTexto")) &&
(pPliego = (BDCPLIEGO) GetProcAddress (*hDll, "BdcPliego")) &&
(pComentario = (BDCCOMENTARIO) GetProcAddress (*hDll, "BdcComentario")) &&
(pClaves = (BDCCLAVES) GetProcAddress (*hDll, "BdcClaves")) &&
(pError = (BDCERROR) GetProcAddress (*hDll, "BdcError")) ) {

*Err = "";
return TRUE;

} else {
*Err = "FUNCION INEXISTENTE EN LA LIBRERIA";
return FALSE;
}
}

BOOL
CierraDll (
HMODULE hDll,
char **Err )
{
if (hDll) {
if ( !FreeLibrary (hDll) ) {
*Err = "ERROR AL CERRAR LA LIBRERIA";
return FALSE;
} else {
*Err = "";
return TRUE;
}
```

```
} else {
*Err = "NO HAY LIBRERIA QUE CERRAR";
return FALSE;
}
}

BOOL
ChequeaError (
HANDLE Precio,
FILE *f,
BOOL EsGlobal )
{
char *MensajeError;
BOOL CodigoError;

CodigoError= EsGlobal ? pGloError (&MensajeError) : pError (Precio, &MensajeError);

if (CodigoError & BDCERR_CORRECTO)
return TRUE;
if (CodigoError & BDCERR_BASE_DATOS)
fprintf (f,"ERROR: %s\n",MensajeError);
if (CodigoError & BDCERR_PARAMETRO)
fprintf (f,"ERROR: El parámetro no existe\n");
if (CodigoError & BDCERR_OPCION)
fprintf (f,"ERROR: La opción no existe\n");
if (CodigoError & BDCERR_MAX_OPCIONES)
fprintf (f,"ERROR: Se alcanzó el máximo número de opciones (62)\n");
if (CodigoError & BDCERR_NO_LEIDO)
fprintf (f,"ERROR: El concepto paramétrico no fue leído\n");
if (CodigoError & BDCERR_NO_CALCULADO)
fprintf (f,"ERROR: El derivado paramétrico no fue calculado\n");
if (CodigoError & BDCERR_DESCOMPOSICION)
fprintf (f,"ERROR: No existe un elemento de la descomposición\n");
if (CodigoError & BDCERR_SIN_CODIGO)
fprintf (f,"ERROR: El concepto no tiene CODIGO definido\n");
if (CodigoError & BDCERR_SIN_MEMORIA)
fprintf (f,"ERROR: Memoria insuficiente\n");
if (CodigoError & BDCERR_CONCEPTO_NULO)
fprintf (f,"ERROR: El concepto es erróneo\n");
```

```
return FALSE;
}
```

2.1.3. Archivo ?programa.c?

```
#define STRICT
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "programa.h"

int main (void) {

HANDLE Precio=(HANDLE)0, PrecioTmp=(HANDLE)0;
HINSTANCE hLib=(HINSTANCE)0;
FILE *fSalida;
char *err, codigo[20];
const char *texto;
char *Precios [] = {"ABPH.1$", "SBRG.1$", "EADR.3$", "NoExiste"};
int nPrecios = 4;
int i,j,k;
long limite,nParGlobales=0,ListaOpcGlobales[20];
long nPar,nOpc[20],ListaOpc[20],ListaOpc2[20];
double numero;

printf ("\nEJEMPLO DE PROGRAMA QUE ACCEDE A LA BASE \"BASE.DLL\"\n");
printf ("LOS RESULTADOS SE ESCRIBEN EN EL ARCHIVO \"SALIDA.TXT\"\n");

if ( !(fSalida = fopen ("SALIDA.TXT", "wt")) ) {
printf ("\nERROR: Imposible crear el archivo de salida\n");
printf ("\n\nPROGRAMA FINALIZADO\nPulse una tecla\n");
getch();
return FALSE;
}
fprintf (fSalida,"\nEJEMPLO DE PROGRAMA QUE ACCEDE A LA BASE \"BASE.DLL\"\n");

if (!AbreDll("BASE.DLL", &hLib, &err)) {
fprintf (fSalida,"\nERROR: %s\n", err);
printf ("\n\nPROGRAMA FINALIZADO\nPulse una tecla\n");
getch();
```

```
return FALSE;
}

// Lectura del Paramétrico Global
printf ("\nBUSCANDO EL PARAMETRICO GLOBAL...");
if ( !(ChequeaError ((HANDLE)0, fSalida, TRUE)) ) {
printf ("\nLEYENDO EL PARAMETRICO GLOBAL...");
nParGlobales = pGloParNumero();
if (nParGlobales==ERRONEO)
ChequeaError ((HANDLE)0, fSalida, TRUE);
fprintf (fSalida,"\nPrecio Global: Número de parámetros: %ld\n", nParGlobales);
for (j=0; j<nParGlobales; j++) {
ListaOpcGlobales[j]=0;
if (!(texto=pGloParRotulo(j)))
ChequeaError ((HANDLE)0, fSalida, TRUE);
else
fprintf (fSalida,"\nParámetro %d: Rótulo: %s\n", j+1, texto);
limite=pGloOpcNumero(j);
if (limite==ERRONEO)
ChequeaError ((HANDLE)0, fSalida, TRUE);
for (k=0; k<limite; k++) {
if (!(texto=pGloOpcRotulo(j,k)))
ChequeaError ((HANDLE)0, fSalida, TRUE);
else
fprintf (fSalida,"\tOpción %d: Rótulo: %s\n", k, texto);
}
}
pGloCalcula (ListaOpcGlobales);
}
// Lectura de los precios
for (i=0; i<nPrecios; i++) {
printf ("\n\nBUSCANDO EL PRECIO \"%s\"...", Precios[i]);
if ( (Precio = pLee (Precios[i])) == (HANDLE) 0 ) {
fprintf (fSalida,"\nERROR: El precio \"%s\" no existe\n", Precios[i]);
continue;
}

// Rótulos de Parámetros y Opciones
printf ("\nLEYENDO EL PRECIO \"%s\"...", Precios[i]);
```

```
nPar = pParNumero(Precio);
if (nPar==ERRONEO)
ChequeaError (Precio, fSalida, FALSE);
else
fprintf (fSalida, "\nPrecio \"%s\": Número de parámetros: %ld\n",
Precios[i], nPar);
for (j=0; j<nPar; j++) {
if (!(texto=pParRotulo(Precio,j)))
ChequeaError (Precio, fSalida, FALSE);
else
fprintf (fSalida,"\nParámetro %d: Rótulo: %s\n", j+1, texto);
nOpc[j]=pOpcNumero(Precio,j);
if (nOpc[j]==ERRONEO)
ChequeaError (Precio, fSalida, FALSE);
ListaOpc[j]=0;
for (k=0; k<nOpc[j]; k++) {
if (!(texto=pOpcRotulo(Precio,j,k)))
ChequeaError (Precio, fSalida, FALSE);
else
fprintf (fSalida,"\tOpción %d: Rótulo: %s\n", k+1, texto);
}
}

//Listado de todas las combinaciones paramétricas
printf ("\nCALCULANDO EL PRECIO \"%s\"...", Precios[i]);
do {
if (pValida (Precio, ListaOpc)==ERRONEO) {
fprintf (fSalida,"\n");
ChequeaError (Precio, fSalida, FALSE);
fprintf (fSalida,"(CODIGO : %s)\n", pCodigo(Precio));
goto siguiente;
}
pCalcula (Precio, ListaOpc);
if (!(texto=pCodigo(Precio))) {
ChequeaError (Precio, fSalida, FALSE);
sprintf (codigo,"");
} else
strcpy (codigo, texto);
if ((PrecioTmp=pDecodifica (codigo, ListaOpc2)) == (HANDLE)0)
```

```
ChequeaError (Precio, fSalida, FALSE);
pCierra (PrecioTmp);
fprintf (fSalida,"\nCODIGO: %s\n", codigo);
fprintf (fSalida,"OPCIONES: ");
for (j=0; j<nPar; j++)
fprintf (fSalida,"%d ",ListaOpc2[j]);
if (!(texto=pResumen(Precio)))
ChequeaError (Precio, fSalida, FALSE);
else
fprintf (fSalida,"\nRESUMEN:%s\n", texto);
if (!(texto=pTexto(Precio)))
ChequeaError (Precio, fSalida, FALSE);
else
fprintf (fSalida,"TEXTO:%s\n", texto);
if (!(texto=pPliego(Precio)))
ChequeaError (Precio, fSalida, FALSE);
else
fprintf (fSalida,"PLIEGO:%s\n", texto);

limite=pDesNumero(Precio);
if (limite==ERRONEO)
ChequeaError (Precio, fSalida, FALSE);
else if (limite) {
fprintf (fSalida,"DESCOMPOSICION:\n");
for (j=0; j<limite; j++) {
if (!(texto=pDesCodigo(Precio,j)))
ChequeaError (Precio, fSalida, FALSE);
else if (pRendimiento(Precio,j,&numero)==ERRONEO)
ChequeaError (Precio, fSalida, FALSE);
else
fprintf (fSalida,"\t%2d %-10s \t%10.3lf\n", j+1, texto, numero);
}
} else {
if (pPrecio(Precio, &numero)==ERRONEO)
ChequeaError (Precio, fSalida, FALSE);
else
fprintf (fSalida,"PRECIO:%.2lf\n", numero);
}
```

```
siguiente:
for (j=nPar-1; j>=0; j--) {
if (ListaOpc[j]<nOpc[j]-1) {
ListaOpc[j]++;
break;
} else {
ListaOpc[j]=0;
}
}
if (j<0) break;
} while (TRUE);

pCierra (Precio);
}

//Ejemplo de decodificar
printf ("\n\nDECODIFICANDO EL PRECIO \"SBRG.1_38\"...");
if ((Precio=pDecodifica("SBRG.1_38",ListaOpc2))==(HANDLE)0)
ChequeaError (Precio, fSalida, FALSE);
else {
if (!(texto=pCodigo(Precio))) {
ChequeaError (Precio, fSalida, FALSE);
sprintf (codigo,"");
} else
strcpy (codigo, texto);
nPar = pParNumero(Precio);
if (nPar==ERRONEO)
ChequeaError (Precio, fSalida, FALSE);
fprintf (fSalida,"\n\nDecodificando SBRG.1_38:");
fprintf (fSalida,"\nCODIGO DEL CONCEPTO PARAMETRICO: %s\n", codigo);
fprintf (fSalida,"OPCIONES: ");
for (j=0; j<nPar; j++)
fprintf (fSalida,"%d ",ListaOpc2[j]);

}
pCierra (Precio);

fcloseall();
```

```
if (!CierraDll ((HMODULE)hLib, &err) ) {
fprintf (fSalida,"\nERROR: %s\n", err);
printf ("\n\nPROGRAMA FINALIZADO\nPulse una tecla\n");
getch();
return FALSE;
}
printf ("\n\nPROGRAMA FINALIZADO\nPulse una tecla\n");
getch();
return TRUE;
}
```