DEFINITION OF THE STANDARD EXCHANGE FORMAT FOR CONSTRUCTION DATABASES FIEBDC-3/2020.
(entry into force as of 01/01/2021)

PRESENTATION.

The following DEFINITION of the STANDARD CONSTRUCTION Database Exchange Format includes the specifications approved up to Minute 27 of the Standing Technical Committee (STC) inclusive (held on 6 October 2020) and ratified by the ASSEMBLY of the FIEBDC Association. Entry into force on 1 January 2020, as VERSION 3/2020.

To facilitate its reading, new or modified paragraphs with respect to the FIEBDC-3/2016 specification are indicated in blue.

This document is made available to users and companies, on the condition that any computerized implementation of this format must include both input and output data.

This format is intended to cover all the INFORMATION contained in current CONSTRUCTION databases. Not all database developers will need to make use of all the format's possibilities. Likewise, not all measurement and budgeting programs will make use of all the INFORMATION provided.

The format itself also provides for extension. Compatibility between versions will be maintained as far as possible when dealing with any new content planned.

FIEBDC-3 FORMAT. SPECIFICATION.

FIEBDC format, i.e. the STANDARD CONSTRUCTION Database Exchange Format, is intended to reconstruct a database or work base with all the necessary INFORMATION, on hardware and software formats other than those on which the INFORMATION was produced.

The INFORMATION included within a database, work base or certification shall be available in a file in FIEBDC format, with the ".BC3" file extension.

The only limitation to file size will be the maximum allowed by the hardware used for its transport. If a file compressor is used, the decompressor must be included in the same storage medium or a self-extracting format must be used.

The character set to be used in the CODE fields shall be that defined by MS-DOS 6.0, including A-Z, a-z, 0-9, ñ, Ñ, < . > (ASCII-46), < $ > (ASCII-36), < # > (ASCII-35), < %> (ASCII-37), < & > (ASCII-38), < _ > (ASCII-95), excluding any other characters such as space or tab, etc.

The end of line shall be the STANDARD for MS-DOS files (ASCII-13 and ASCII-10). The end of file shall be marked according to the same STANDARD (ASCII-26). Tab (ASCII-9) is the only additional control character allowed.

Each file shall consist of registry, text areas between the beginning of registry< ~ > (ASCII-126) character and the next beginning of registry or end of file. Files shall contain complete registries, i.e. file splitting shall be performed at the beginning of a registry (< ~ > character).

Each registry shall consist of fields separated by < | > (ASCII-124) characters. Any field with INFORMATION shall end with the field separator. The registry shall contain all field separators mentioned above, even if they do not contain INFORMATION. It is not necessary to have field terminators for fields after the last field with INFORMATION.

Each field shall consist of subfields separated by < \ > (ASCII-92) characters. The final separator between the last entry in a field and the end of the field is optional.

The first field of each registry is the registry header, a capital letter identifying the type of registry.

Any INFORMATION between the last field separator of a registry (< | > character) or the beginning of the file and the beginning of the next registry (< ~> character) shall be ignored.

White (32), tab (9) and end-of-line (13 and 10) characters in front of separators < ~ >, < | > and < \ > shall be ignored.

Second order fields (subfields) cannot be partially maintained. The complete INFORMATION of a field in any of the registries shall be updated.

The arrangement of registries within a file is completely optional. However, the sequential reading of registries shall be ensured to avoid discrepancies in the substitution of INFORMATION.

Empty fields will be considered as having NO INFORMATION, rather than containing null INFORMATION. This allows to produce update files containing only the INFORMATION in one of its fields and, by default, the reference CODE.

To invalidate a numeric field, the value 0 (zero) shall explicitly appear.

To invalidate an alphanumeric field, the NULL LABEL must be explicitly displayed.

To collect several certifications at the same time, the name of the file containing actual cost must have the same name as that of the budget, adding (concatenating) "#certification NNNN" where NNN would be the certification number. It is recommended to place the file in the same folder as the budget folder. This will allow a budget and all its certifications (or only those selected) to be imported at the same time. The file containing the certification is identical to that of a budget and differs only in the ~V registry, which will contain the certification and the certification number and date. It contains all information (~D, ~M, ~C, ~T), and not simply the registries of the certified units (~D, ~M).

In order to enable the transfer from Internet databases to applications on local computers using FIEBDC, it is indicated that when dragging and dropping the FIE icon on an application, the application in the drop event receives a URL ending with id_concept, with the extension .bc3, to which the program will have to make a request that will returns a .bc3 file.

NOTATION CONVENTION.

[a]	Indicates nothing or "a".
{a}	Indicates zero or more "a" occurrences.
<a>	Indicates one or more "a" occurrences.
(<DD>c)	Maximum size in number of field characters.

All numeric values must be entered without thousand separators and with the full-stop character “.” between the integer and decimal parts.

OWNERSHIP AND VERSION-TYPE REGISTRY.

This registry is used to document the file source and format and, if available, shall be provided at the beginning of the first file.

~V | [FILE_OWNERSHIP] | [FORMAT_VERSION] [\ DDMMYYYY] | [EMISSION_PROGRAM] | [HEADER] \ { IDENTIFICATION_LABEL \ } | [CHARACTER_SET] | [COMMENT] | [INFORMATION TYPE] | [CERTIFICATION NUMBER] | [CERTIFICATION DATE] | [URL_BASE] |

FILE_OWNERSHIP: Database editor or work, date...

FORMAT_VERSION: VERSION of the file format. Current version is FIEBDC-3/2016.

DDMMYYYY: DD represents the day with two digits, MM the month and YYYY the year. If the date has six digits or less, the year will be displayed with two digits (YY), interpreted using the "80/20" criterion. In other words, any year that is equal to or greater than 80 will correspond to the twentieth century and any year that is less than 80 will correspond to the twenty-first century. If the date has less than five digits it represents only the month and year (MMYY), if it has less than three digits, it only represents the year (YY). If the date is identified by an odd number of digits, it shall be completed with the character zero on the left. To represent a date without a specific day or month, a double zero shall be used in each case.

Examples:

12062000	12 June 2000.
120699	12 June 1999.
00061281	June 1281.
061281	6 December 1981.
401	April 2001.

EMISSION_PROGRAM: Either the program or the company that generates the files in BC3 format.

HEADER: General title of the IDENTIFICATION_LABEL.

IDENTIFICATION_LABEL: This sequentially assigns titles to the values defined in the PRICE field of the ~C registry, and the decimal number field sets of the ~K registry, which, as indicated in its SPECIFICATION, may represent different time periods or geographic areas, etc., establishing a two-way relationship between these. See Annexes 5 (Territorial areas) and 6 (Currencies).
If there are more IDENTIFICATION_LABEL fields in the ~V registry than PRICE fields in the ~C registry, or than more than the number of sets of decimal fields in the ~K registry, it shall be understood that the PRICE and the sets of decimal fields of this remainder shall be equal to the last one defined.

CHARACTER_SET: Indicates whether the character set to be used is the set defined for D.O.S., whose identifiers will be 850 or 437, or the set defined for Windows, whose identifier will be ANSI. If this field is empty, it will be understood, by default, that the character set to be used will be 850 for compatibility with previous versions.

COMMENT: File contents (base, work., etc.).

INFORMATION TYPE: Index of the type of information to be exchanged.
The following types are defined:

1	Database.
2	Budget.
3	Actual cost.
4	Database update.

CERTIFICATION NUMBER: Numeric value stating the certification order (first, second, third, etc.) This is only relevant when the information type is Certification.

CERTIFICATION DATE: Date of the certification specified in the certification number field. This is only relevant when the information type is Certification. The date shall be defined in the same format as the DDMMYYYY field of this registry.

URL_BASE: URL from which the documents and graphics will be found.

COEFFICIENT-TYPE REGISTRY.

This indicates the number of decimals in each numeric field. When the numeric field has a negative sign, it indicates the maximum number of decimals. Otherwise, it indicates the exact number of decimals.

In a FIEBDC file, when a quantity appears with more decimal digits than the corresponding numbers according to the ~K registry, it must be specified as an error. This is because that which is exported must match the ~K registry.

When this registry does not exist in a FIEBDC file, it is recommended to apply the default decimals set in the format.

~K | { DN \ DD \ DS \ DR \ DI \ DP \ DC \ DM \ CURRENCY \ } | CI \ GG \ BI \ REDUCTION \ VAT | { DRC \ DC \ \ DFS \ DRS \ \ DUO \ DI \ DES \ DN \ DD \ DS \ DSP \ DEC \ CURRENCY \ } | [n] |

This registry includes field 1 for compatibility with previous versions of the format, although the programs should read field 3, as it is more complete and, failing that, field 1.

Preliminary concepts.

Work unit: Any individual or compound element, with or without indirect costs, which is used in a budget.
Compound element: Any construction element that contains a decomposition and is neither a root nor a chapter.
Individual element: Any construction element that does not contain a decomposition and is neither a root nor a chapter.

Definitions.

DN	Decimals of the equal-parts number field of the measurement sheet. Two decimals by default.
DD	Decimals of dimensions of the three magnitudes of the measurement sheet. Two decimals by default.
DS	Decimals of the subtotal or total measurement line. Two decimals by default.
DR	Decimals of output and factor in a decomposition. Three decimals by default.
DI	Decimals of the amount resulting from multiplying output and price of the concept. Two decimals by default.
DP	Decimals of the amount resulting from the sum of the direct costs of the concept. Two decimals by default.
DC	Decimals of the total amount of the concept. (CD+CI). Two decimals by default.
DM	Decimals of the amount resulting from multiplying the total measurement of the concept by its price. Two decimals by default.
CURRENCY	Currency expressed using the same abbreviations as those used by the ECB (European Central Bank), which, where appropriate, must coincide with those of the ~V registry. Current abbreviations are shown in Annex 6.
CI	Indirect costs, expressed as a percentage.

GG	General Company Overheads, expressed as a percentage.
BI	Contractor's Industrial Profit, expressed as a percentage.
REDUCTION	Coefficient of decrease or increase of an award budget, expressed as a percentage.
VAT	Value Added Tax, expressed as a percentage.
DRC	Output and output-factor decimals of a budget, and decimals of the result of their multiplication. Three decimals by default.
DC	Decimals of the amount of an estimate, its chapters, subchapters, etc. and measurement lines (excluding work units), and decimals of the amounts resulting from multiplying the total output (or measurement) of the estimate, its chapters, subchapters, etc. and measurement lines (excluding work units) by their respective prices. Two decimals by default.
DFS	Decimals of the output factors of the work units and compound elements. Three decimals by default.
DRS	Decimals of the output of the work units and compound elements and decimals of the result of the multiplication of these outputs by their respective factors. Three decimals by default.
DUO	Decimals of the total cost of the work units. Two decimals by default.
DI	Decimals of the amounts resulting from multiplying the total outputs of either compound elements or individual elements by their respective prices, decimals of the amount resulting from the sums of the direct costs of the work unit and decimals of the indirect costs. Decimals of the sums to which the percentages are applied. Two decimals by default.
DES	Decimals of the amount of the individual elements. Two decimals by default.
DN	Decimals of the equal-parts number field of the measurement sheet. Two decimals by default.
DD	Decimals of dimensions of the three magnitudes of the measurement sheet. Two decimals by default.
DS	Decimals of the subtotal or total measurement line. Two decimals by default.
DSP	Decimals of the measurement subtotal line. Two decimals by default.
DEC	Decimals of the amount of compound elements. Two decimals by default.
CURRENCY	Currency expressed using the same abbreviations as those used by the ECB (European Central Bank), which, where appropriate, must coincide with those of the ~V registry. Current abbreviations are shown in Annex 6.
n	Is the number of the option of the BdcGloParNumero function that refers to the currency concept.

If an amount has a greater number of decimal digits than those specified in the ~K registry, it must be rounded to the number of decimal digits indicated in the ~K registry (criterion <5 remains the same and >= 5 adds up), and the operations carried out on the amount will use this rounded value.

To relate a certain currency with its decimal convention, the order in which the different currencies appear in the ~K registry must coincide with the order indicated in the ~V registry or with the option order of the BdcGloOpcNumero function, depending on the case. If in the ~V registry, there are more LABELS than currencies in the ~K registry, the prices without their corresponding currency will take the last currency defined in the ~K registry.

Example:

Example of a price base providing amounts for two territorial areas in two currencies (the fields concerned are indicated in bold):

> ~V | FILE_OWNERSHIP | FORMAT_VERSION \ DDMMYY | EMISSION_PROGRAM | [**Prices of different territorial areas in different currencies**] \ { **B-eur \ T-eur \ B-usd \ T-usd** } | CHARACTER_SET | [INFORMATION TYPE] | [CERTIFICATION NUMBER] | [CERTIFICATION DATE] |

~C | CODE { \ } | UNIT | SUMMARY | { **120** \ **108** \ **102.8** \ **92.52** } | { DATE \ } | TYPE |

~K | { DN \ DD \ DS \ DR \ DI \ DP \ DC \ DM \ CURRENCY \ } | CI \ GG \ BI \ REDUCTION \ VAT | { DRC \ DC \ \ DFS \ DRS \ \ DUO \ DI \ DES \ DN \ DD \ DS \ DSP \ DEC \ **eur** \ DRC \ DC \ \ DFS \ DRS \ \ DUO \ DI \ DES \ DN \ DD \ DS \ DSP \ DEC \ **usd** \ } | [n] |

Explanatory diagram.

Concept	***Amount***
Root chapter (##)	DC (2)

Concept intervening in a decomposition line:

Concept	***Factor***	***Output***	***OF*** Factor*Outp	***Price***	***Amount*** OF*Price
Unitary budget	DRC (3)	DRC(3)	DRC(3)	DC(2)	DC(2)
Chapter (#)					DC(2)*
Work unit	DFS(3)	DS(2)	DS(2)	DUO(2)	DC(2)
Compound element	DFS(3)	DRS(3)	DRS(3)	DEC(2)	DI(2)
Individual element	DFS(3)	DRS(3)	DRS(3)	DES(2)	DI(2)

* The Chapter amount (#) shall be the sum of the amounts of its components.

Calculation of the price of a work unit:

Work unit	**Direct cost**	**Indirect cost**	***Price*** Direct cost + Indirect cost
Compound element	DI(2)	DI(2)	DUO(2)
Individual element	DI(2)	DI(2)	DUO(2)

Measurements	***Quantity***	***Dimensions***	***Subtotal***	***Total***
Measurement lines	DN(2)	DD(2)	DSP(2)	DS(2)

The default decimals have been added according to the format after each term.

CONCEPT-TYPE REGISTRY.

This registry includes the basic INFORMATION of a concept of any type, material, auxiliary, item, chapter, entity, document, etc., both in its parametric VERSION and traditional DEFINITION.

~C | CODE { \ CODE } | [UNIT] | [SUMMARY] | { PRICE \ } | { DATE \ } | [TYPE] |

CODE: CODE of the described concept. A concept can have several CODES, which will be synonymous. This mechanism allows the integration of different classification systems. It can have a maximum of 20 characters.

To differentiate the root-type concept of a file, as well as the chapter-type concepts, its CODE shall be extended with the characters '##' and '#' respectively. This NOTATION shall be compulsorily shown in the ~C type registry, and is optional in the remaining registries of the same concept.
References to a CODE with and without either # or ##, are understood to be unique to the same concept.

There can only be one root concept in a database or work.

If a code has a '#' character inserted, it shall be understood that it corresponds to the set ENTITY_CODE # CONCEPT_CODE defined in the Business-Relationship Type registry (~O registry) or in the function BdcComercCodigo.

UNIT: Unit of measurement. There is a list of recommended units of measurement, developed by the Asociación de Redactores de Bases de Datos de Construcción. See Annex 7 on Units of Measurement.

SUMMARY: Summary of the descriptive text. Each format shall indicate the number of characters allowed in its summary field. A maximum of 64 characters is recommended.

PRICE: Price of the concept. A concept may have several alternative prices that represent different periods or geographical scopes, etc., defined biunivocal with respect to the field [HEADER {IDENTIFICATION_LABEL} of the ~V registry. When there is more than one price, they shall be assigned sequentially to each defined LABEL; if there are more LABEL than prices, the last price defined shall be assigned to them. If the concept has a decomposition, this price will be the result of said decomposition. This price must be provided to allow its verification. In case of discrepancy, the result obtained by the decomposition, as indicated in the Decomposition type registry, ~D, shall have precedence, and the user may be informed of this situation. This also applies to chapter and root concepts of a Work or Budget. An exception to this rule is the exchange of unstructured measurements (see description of the Measurements-type registry, ~M).

DATE: Date of last price update. When there is more than one date, they will be assigned sequentially to each defined price. If there are more prices than dates, the prices without their corresponding date will take the last defined date.

Dates shall be defined in the format DDMMYYYY: DD represents the day with two digits, MM the month and YYYYY the year. If the date has six digits or less, the year will be displayed with two digits (YY), interpreted using the "80/20" criterion. In other words, any year that is equal to or greater than 80 will correspond to the twentieth century and any year that is less than 80 will correspond to the twenty-first century. If the date has less than five digits it represents only the month and year (MMYY), if it has less than three digits, it only represents the year (YY). If the date is identified by an odd number of digits, it shall be completed with the character zero from the left. To represent a date without a specific day or month, a double zero shall be used in each case.

Examples:

12062000	12 June 2000
120699	12 June 1999
00061281	June 1281
061281	6 December 1981
401	April 2001

TYPE: Concept type. Initially the following types are reserved:

0	Unclassified
1	Labour
2	Machinery and auxiliary equipment
3	Materials
4	Additional waste components
5	Waste classification

The use of the classification indicated by the Boletín Oficial del Estado (BOE) and the CNC is also permitted (and recommended) in indexes and polynomial formulas for price revision as well as those recommended by the Asociación de Redactores de Bases de Datos de Construcción. The rates currently in force are shown in Annex 4.

DECOMPOSITION-TYPE REGISTRY.

This registry includes the decomposition of a concept into other concepts by means of one or two quantities. The same registry is used to define the decomposition of a work-unit type concept into material, labour, machinery, and auxiliary-type concepts and for the decomposition of a chapter-type concept into work-unit or sub-chapter type concepts.

If a derived concept is defined as a concept that does not have variable parts in its definition (it is not a parametric) nor is it a chapter, then a derived concept can only contain derived concepts in its decomposition.

~D | PARENT_CODE | < CHILD_CODE \ [FACTOR] \ [OUTPUT] \ > | < CHILD_CODE \ [FACTOR] \ [OUTPUT] \ {PERCENTAGE_CODE ; } \ > |

This registry includes field 2 for compatibility with previous versions of the format, although the programs must read field 3 as it is more complete and, failing that, field 2.

PARENT_CODE: CODE of the decomposed concept.

CHILD_CODE: CODE of each concept involved in the decomposition.

FACTOR: Output factor. One by default.

OUTPUT: Number of units, output, or measurement. One by default.

When a chapter is included in a decomposition line, it is not affected by either the factor or the output. These fields are exchanged with the default value one in that decomposition line.

When the CHILD_CODE includes the "%" or the "&" character, it is a percentage over the previous lines of the decomposition. The percent code has three parts:

1) Prefix, which forms a mask indicating to which elements the percentage is applied. If the prefix is null, the percentage is applied to all preceding lines.

2) A character, which may be "&" (cumulative percentage), or "%" (non-cumulative percentage).

3) A series of free characters to differentiate one percentage from another.

Example: OP%N0001
OP: On all previous lines whose code begins with OP.
%: Non-cumulative percentage
N0001: Differentiating code.
The output will be the percentage that is applied on the lines before the current one and which are affected by the mask.

Example of a decomposition line: O%N0001 \\0.03\

This line represents a percentage of 0.03 per unit (3%) of all lines before the current line, including percentages, whose code begins with O and whose text will be in the definition of the code “O%N0001”.

Example: ~C | O%N0001 | % | Auxiliary equipment |

For the purposes of calculating compound prices, cumulative and non-cumulative percentages have the same behaviour. The difference between them only appears in the calculation of simple quantities in a budget, for which the cumulative ("&") will be considered as percentages of losses, breakages or other cases that imply a greater quantity of resources in higher lines. The non-cumulative (“%”) may refer to small material or other cases that do not imply a greater need for resources in higher lines.

The existence of the factor in decomposition lines and the almost non-existent use that has been made of the cumulative percentages (“&”) means that these are maintained for historical reasons, but their use is not recommended.

PERCENTAGE_CODE: Percentage concept CODES applied to this decomposition line. This has free coding as the code is not used to determine to which lines the percentage is applied separated with the character <; > (ASCII-59). In a decomposition, the code percentage expressly appears in the lines on which it is applied: these will univocally determine the application of the percentage and another line with the code percentage must appear in the field CHILD_CODE in the decomposition. In this case, the rules on the percentage code composition do not apply, but those used for the remaining concepts do. The percentage will be applied to the sum of the lines preceding it and bearing the PERCENTAGE_CODE. A percentage concept cannot appear in the PERCENTAGE_CODE position after it appears in the CHILD_CODE. More than one percentage concept may be applied on the same line.

If the amount of a concept can be obtained through the price field of the ~C registry and through the ~D registry, the amount obtained from the latter shall take precedence over the amount of the former.

ADD-DECOMPOSITION TYPE REGISTRY.

This registry can be used to add decomposition lines, while the ~D-type registry changes the entire decomposition. To add new concepts to a database, in addition to defining the ~C, ~T, ~L, ~D registries, etc., the new concepts should be inserted in the chapter or chapters in which the user wishes to place them. For this purpose, a registry allowing the user to add one or several decomposition lines for each chapter in which the user wishes to place a new concept is required.

~Y | PARENT_CODE | < CHILD_CODE \ [FACTOR] \ [OUTPUT] \ > | < CHILD_CODE \ [FACTOR] \ [OUTPUT] \ { PERCENTAGE_CODE ; } \ > |

All fields have the same meaning as in the ~D registry type.

WASTE DECOMPOSITION TYPE REGISTRY.

This registry is used to assign the components that generate waste to a concept and, furthermore, to provide properties dependent on the parent-child relationship.

~ R | PARENT_CODE | { DECOMPOSITION_TYPE \ CHILD_CODE \ { PROPERTY \ VALUE \ [UM] \ } | } |

PARENT_CODE: Code of the concept, contained in the database.

DECOMPOSITION_TYPE: Type of components into which a concept can be decomposed. Initially the following are defined:

0 Placement-component waste.
1 Demolition-component waste.
2 Excavation-component wastes.
3 Packaging-component wastes.

CHILD_CODE: Code of the concept that intervenes as a component of PARENT_CODE in the DECOMPOSITION_TYPE relation, contained in the database.

PROPERTY: Technical information of CHILD_CODE related to PARENT_CODE. When calculating the waste, two properties will be used:

- Output.

wf Waste factor. Total percentage of placement waste. wf will be used when CHILD_CODE corresponds to placement waste.

VALUE: Alphabetical or numerical value of the property.

UM: Unit of measurement. In case the values of the property are numerical, it shall be indicated according to the International System of Units of Measurement (see Annex 7).

Waste calculation.

When calculating waste in concepts described as discrete, ~R and ~X registries will be used. For concepts described with complex parametric descriptions in DLLs, BdcNumProp, BdcPropValString, BdcNumComponent, BdcComponentCode, BdcNumPropComponent and BdcComponentPropValString will be used.

An item (material, unitary element, etc.) can generate one or more wastes.
Each waste must have a defined EWC classification, volume (m3) and mass (kg).
In addition to EWC, a waste can also be classified by other classification systems.
The waste obtained is calculated in the element's unit of measurement.

Waste of an individual element.

The waste of an individual element is obtained from direct values obtained from the IT_CODE field of the ~X registry (properties: ewc, v, m).

Where,

EWC of an element's waste.

EWC classification of the waste (European Waste Catalogue, according to Order MAM/304/2002).
The EWC of an element's waste can be either a direct value chosen from the EWC list or the cumulative list of EWC values obtained from its components. For individual elements the EWC is always a direct value.

[]. ewc = direct value, or
if []. ewc = null, [ ].ewc = EWC accumulation of its components

Volume of an element's waste.

Volume occupied by the element in space.
Volume can be a direct value, or it can be calculated from the sum of the volume of each component, times the amount of each component in the element (Qobra). For individual elements the volume is always a direct value.
Calculation of the volume of an element (m3 per ua, unit of measurement):

[]. v = direct value, or
if []. v = null, [ ].v = sum ([C].v * [ ].[C].Qobra (ua))
Where,
- C = component
- Qobra (amount of each component in the element) = Output*Factor - Waste

Mass of an element's waste.

The mass can either be a direct value or calculated from the sum of the mass of each component times the amount of each component in the element (Qobra). For individual elements the mass is always a direct value.
Calculation of the mass of an element (kg per ua, unit of measurement):

[].m = direct value, or
if [].m = null, [ ].m = sum ([C].m * [ ].[C].Qobra (ua))

Where,
- C = component
- Qobra = Output*Factor - Waste

Compound-element waste.

The waste of a compound element is obtained from the ~R registry from the sum of the waste of its “waste components”.

The initially defined “waste components” are the following:

- “Placement components”: The waste is the material that is thrown away in the process of work formation.
  These components must exist in the ~D registry.
  They are typified in the ~C registry with TYPE: 0, 1, 2 or 3.
  To calculate the waste of a "placement component" (C), apart from the "Output" and "Factor", which are obtained from the ~D registry, a new variable is needed, the "Waste Factor" (wf), which, if it exists is obtained from the ~R registry.
  The waste of a "placement component" (C) is:
  []. [C].Waste= [ ].[C].Output*Factor * [ ].[C].WasteFactor
  Where, Waste Factor is the total percentage of placement waste.

- “Additional waste components”:
  - Demolition components: The waste is that which is generated from the demolition activity of a constructive element.
  - Excavation components: The waste is that which is generated from the excavation activity of a constructive element.
  - Packaging components: The waste generated from the packaging in which the materials of a construction element are wrapped.

  These components are not included in the ~D registry.
  They are typified in the ~C registry with TYPE: 4.
  To calculate the waste of a “waste component” (R), its output is needed, which is obtained from the ~R registry.
  The waste of an “additional residue component” (R) is:
  []. [R].Waste= [ ].[R].Output
  In other words, the waste of an "additional waste component" (R) in an element is the amount of the packaging, excavation, and demolition components in the element.

Example diagram of the calculation of compound-element waste:

Parent: "Demolition of wall and subsequent construction of brick wall with mortar..."

Child1: “Demolition 1 …….” (~C type 4) = EWC (010407)
Unit mass (7) * output (3) = mass
Volume unit (20) * output (3) = volume
Child2: “Demolition 2 …….” (~C type 4) = EWC (010502)
Unit mass (8) * output (1.5) = mass
Volume unit (21) * output (1.5) = volume
Child3: "Brick ....." (~C type 3) = EWC (010301)
Unit mass (9) * output * factor * wf (1.3) = mass
Volume unit (22) * output * factor * wf (1.3) = volume
Child4: "Mortar ....." (~C type 3) = EWC (010702)
Unit mass (10) * output * factor * wf (1.5) = mass
Volume unit (23) * output * yield * factor * wf (1.5) = volume
Child5: "Packaging 1 ....." (~C type 4) = EWC (010801)
Unit mass (11) * output (7) = mass
Volume unit (24) * output (7) = volume

Child6: “Packaging 2 .....” (~C type 4) = EWC (010903)
Mass unit (12) * output (8.2) = mass
Volume unit (25) * output (8.2) = volume

Where the values "EWC", "unit mass" and "unit volume" are obtained from the ~X registry. "Children", "wf" and “outputs" of the ~C type 4 are obtained from the ~R registry.

Its expression would be:

~R | Parent | 1 \ Child1 \ o \ 3 \ \ | 1 \ Child2 \ o \ 1.5 \ \ | 0 \ Child3 \ wf \ 1.3 \ \ | 0 \ Child4 \ wf \ 1.5 \ \ | 3 \ Child5 \ o \ 7 \ \ | 3 \ Child6 \ o \ 8.2 \ \ | |

~X | Child1 | ewc \ 010407 \ m \ 7 \ v \ 20 \ |
~X | Child2 | ewc \ 010502 \ m \ 8 \ v \ 21 \ |
~X | Child3 | ewc \ 010301 \ m \ 9 \ v \ 22 \ |
~X | Child4 | ewc \ 010702 \ m \ 10 \ v \ 23 \ |
~X | Child5 | ewc \ 010801 \ m \ 11\ v \ 24 \ |
~X | Child6 | ewc \ 010903 \ m \ 12 \ v \ 25 \ |

Waste classification.

Apart from EWC, waste can also be classified by other classification systems. For example:
- Type of waste, according to Directive 1999/31/EC: inert, non-hazardous, hazardous.
- Minimum fractions, according to Real Decreto 105/2008: concrete, roof tiles and ceramic materials, wood, plastic, paper, and cardboard packaging, etc.

Once the waste’s EWC has been obtained, the ~C and ~D registries are used to obtain its classification according to other categories. To identify the categories and the concepts included in these categories within the ~C registry, the TYPE of concept will be used: 5. For each classification this value is totalized having been converted into m3 and kg.

ENTITY-TYPE REGISTRY.

This registry contains the descriptive text of a concept

~T | CONCEPT_CODE | DESCRIPTIVE_TEXT |

CONCEPT_CODE: CODE of the described concept.

DESCRIPTIVE_CODE: Descriptive text of the concept with no size limitation. The text may contain end-of-line characters (ASCII-13 + ASCII-10) which will be maintained when reformatted.

PARAMETRIC-DESCRIPTION TYPE REGISTRY.

This registry contains the parametric description, either in traditional format or in API format for DLLs, which includes the DEFINITION of parameters, decompositions, comments to help in the selection of parameters, summaries, texts, specifications, keys, and commercial INFORMATION, in terms of tables, expressions and variables, for a family of concepts.

This registry can be in two forms:

~P | | [PARAMETRIC_DESCRIPTION] | [DLL_NAME] |

When FAMILY_CODE is full, PARAMETRIC_DESCRIPTION will either be full or empty. In the latter case the parametric description of the family is accessed through the file NAME.DLL.

~P | FAMILY_CODE | [PARAMETRIC_DESCRIPTION] |

When FAMILY_CODE is empty, it refers to the global parametric.

If PARAMETRIC_DESCRIPTION is full, the global parameter is established therefrom. If PARAMETRIC_DESCRIPTION is empty and DLL_NAME is full, it is established from the latter. If PARAMETRIC_DESCRIPTION and DLL_NAME.DLL are both full, only PARAMETRIC_DESCRIPTION is valid.

FAMILY_CODE: CODE of the family-type concept described. If a parameter-dependent coding model is used (see Annexes 2 and 3), this code must have a "$" character in its seventh position, and the concepts in which it is derived will have as their code the first six characters of said code plus an additional character for each parameter it owns.

PARAMETRIC_DESCRIPTION: See Annex 2.

NAME.DLL: See Annex 3.

SPECIFICATIONS-TYPE REGISTRY.

This registry contains the different sections and texts of the specifications of a concept. The specifications are structured hierarchically using the Coding Classification System and faceted into several sections with different content.

Specification sections.

When the first field of the ~L registry is empty, the registry defines the CODES of the SECTIONS of each specification and their corresponding LABELS. This registry is unique for a database or work.

~L | | < SPECIFICATION_SECTION_CODE \ [SPECIFICATION_SECTION_LABEL] \ > |

SPECIFICATION_SECTION_CODE: CODE defining each SECTION or facet of the specification.

SPECIFICATION_SECTION_LABEL: DEFINITION of the LABEL associated with each corresponding CODE of each SECTION or facet of the specification.

Example of the specifications sections defined for the CONSTRUCTION Database of the Community of Madrid and the CONSTRUCTION Database of the Community of Valencia, indicating CODE and SECTION LABEL:

~L | | DES \ DESCRIPTION AND ADDITIONS TO THE TEXT
\ PRE \ PRE-REQUISITES FOR IMPLEMENTATION
\ COM \ COMPONENTS
\ EJE \ IMPLEMENTATION AND ORGANIZATION
\ NOR \ STANDARDS
\ CON \ CONTROL AND ACCEPTANCE
\ SEG \ SAFETY AND HYGIENE
\ VAL \ VALUATION AND MEASUREMENT CRITERIA
\ MAN \ MAINTENANCE
\ VAR \ MISCELLANEOUS \ |

Model 1 of specifications texts.

When the first field of the ~L registry is not empty, it identifies a given concept. There may be one such registry for each concept in a database or work.

~L | CONCEPT_CODE | { SPECIFICATION_SECTION_CODE \ SPECIFICATION_SECTION_TEXT \ } |
{SPECIFICATION_SECTION_CODE \ TEXT_FILE_RTF \ } | { SPECIFICATION_SECTION_CODE \ TEXT_FILE__HTM \ } |

CONCEPT_CODE: CODE of the concept described, as contained in the database.

SPECIFICATION_SECTION_CODE: DEFINITION of the CODE associated to each specification.

SPECIFICATION_SECTION_TEXT: Text assigned to each facet or SECTION of the concept specification.

The specification for each concept shall be divided by "\" characters into several sections or facets, intended to be printed together or separately.

The line ends of each SECTION of the specification shall be treated as in the TEXT-TYPE REGISTRY.

RTF_TEXT_FILE: This is the name of the file containing the text in RTF format assigned to each SECTION of the concept specification. This file must be in the same directory in which the. BC3 file(s) including its reference are located.

HTM_TEXT_FILE: This is the name of the file containing the text in HTM format assigned to each SECTION of the concept specification. This file must be in the same directory in which the. BC3 file(s) including its reference are located.

Model 2 of specifications texts.

Another option allows to assign the specifications by means of text paragraphs associated to concepts, using the following registry layout, as an alternative to that above:

~Q | < CONCEPT_CODE \ > | {SPECIFICATION_SECTION_CODE \ PARAGRAPH_CODE \ { SCOPE_ABREV ; } \ } |

~J | PARAGRAPH_CODE | [PARAGRAPH_TEXT] | | [PARAGRAPH_FILE_RTF] | [PARAGRAPH_FILE_HTM] |

CONCEPT_CODE: CODE of the concept described, as contained in the database. It will be unique for each ~Q registry.
This registry is for substitution of INFORMATION, it is not for accumulation.

SPECIFICATION_SECTION_CODE: DEFINITION of the CODE associated to each specification. It corresponds to the one defined in the ~L specification header.

PARAGRAPH_CODE: CODE of the text associated to each specification section.

SCOPE_ABREV: Identifier of the geographical scope of the specification section. It is defined in a separate registry.

PARAGRAPH_TEXT: Text that defines the content of the specifications associated to a concept and is identified with PARAGRAPH_CODE.

PARAGRAPH_TEXT_RTF: Text that defines the content of the specifications associated to a concept and is identified with PARAGRAPH_CODE, in RTF format, optionally, with the PARAGRAPH_TEXT remaining mandatory in any case.

PARAGRAPH_FILE_RTF: Name of the file in RTF format that defines the content of the specifications associated to a concept and is identified with PARAGRAPH_CODE. This file must

be in the same directory in which the. BC3 file(s) including its reference are located.

PARAGRAPH_FILE_HTM: Name of the file in HTM format that defines the content of the specifications associated to a concept and is identified with PARAGRAPH_CODE. This file must be in the same directory in which the. BC3 file(s) including its reference are located.

GEOGRAPHIC-SCOPE TYPE REGISTRY.

This establishes the geographic scope corresponding to the Specifications associated to the Database. It does not necessarily have to correspond to the HEADER field defined in the ~V registry.

~W | < SCOPE_ABREV \ [SCOPE] \ > |

SCOPE_ABREV: Abbreviated name that identifies the geographical territory to which it refers. (Autonomous Community, Province, Region, County, Locality, etc.). The identifier < * > (ASCII - 42) indicates GENERAL_SCOPE and represents the entire national territory.

SCOPE: Full name of the geographic territory.

There is a list of recommended abbreviations, elaborated by the Asociación de Redactores de Bases de Datos de Construcción, which can be consulted in Annex 5.

GRAPHIC-INFORMATION TYPE REGISTRY.

This registry contains the graphic file(s) associated with a concept. All external files can be in the same directory in which the. BC3 file(s) including their reference are located or at the url indicated.

~G | CONCEPT_CODE | < GRAPH_FILE.EXT \ > | [URL_EXT] |

CONCEPT_CODE: CODE of the concept described, as contained in the database.

GRAPH_FILE.EXT: Name of the file containing the graphic INFORMATION. Standardized general-use programs will be used as reference, to check and verify the content of the file. These programs will be:

Raster files:	.BMP,. PCX extension:	Microsoft Windows.
	.GIF, .JPG, .PNG extensions:	Microsoft Paint.
	.TIF extension:	Adobe Photoshop.
Vector files:	.WMF extension:	Microsoft Word.
	.DXF extension:	AutoCAD.

URL_EXT: is an optional field. If this is not empty, it is the url to be added to the URL_BASE to find the graphic. URL_BASE is defined in the ~V registry. It behaves as follows: first the graphic is searched for in the local directory and if it is not there, it is searched for in URL_BASE + URL_EXT + FILE_GRAPH.EXT.

ENTITY-TYPE REGISTRY.

It defines the entities supplying technical documentation, price lists and specifications of the concepts contained in the database.

~E | ENTITY_CODE | [SUMMARY] | [NAME] | { [TYPE] \ [SUBNAME] \ [ADDRESS] \ [PC] \ [TOWN] \ [PROVINCE] \ [COUNTRY] \ { TELEPHONE ; } \ { FAX ; } \ { CONTACT_PERSON

; } \ } | [TAX ID] \ [WEB] \ [EMAIL] \ |

ENTITY_CODE: CODE of the SCc that defines the entity (company, organization, etc.).

SUMMARY: Abbreviated name of the entity.

NAME: Full name of the entity.

TYPE: The following are defined:

C	Central.
D	Delegation.
R	Representative.

SUB-NAME: Name of the delegation or representative if different from the central office. It will usually be empty.

ADDRESS \ PC \ TOWN \ PROVINCE \ COUNTRY: Postal address of the entity with all its data, with one address for each type of subfield, in order and sequence.

TELEPHONE: Telephone numbers of the entity, in order and sequence with respect to the subfield type, separated by the character <; > (ASCII-59). It shall be indicated by nine numeric characters, including the province prefix.

FAX: Fax numbers of the entity, with the same specifications as the previous field.

CONTACT_PERSON: Name(s) of the contact person(s) within the entity and position(s) held, there may be several associated with each subfield type, and as such these are separated by the ASCII-59 character.

TAX ID: Company’s tax identification code.

WEB: Company website.

MAIL: Company e-mail address.

COMMERCIAL-RELATIONSHIP TYPE REGISTRY.

This registry establishes the links between the General Database (GDB) concepts with the Specific Database (SDB) commercial products, or vice versa.

Thus, a Database (DB) may contain generic CONCEPTS from a GDB, CONCEPTS referring to commercial products from an SDB, or both at the same time.

~O | DB_ROOT_CODE # CONCEPT_CODE | | < FILE_CODE \ ENTITY_CODE # CONCEPT_CODE \ > |

DB_ROOT_CODE # CONCEPT_CODE: Identifier of a DB concept, where:

DB_ROOT_CODE: Refers to the identification of the CODE of the entity that creates the DB. This CODE must be provided by the entity that prepares the DB, to avoid ambiguities. It is recommended that this be the entity's own VAT number.

CONCEPT_CODE: This refers to a concept that belongs to the DB_ROOT_CODE and is used by the latter in its coding classification system.

FILE_CODE: Refers to the name of the file that, if it exists, indicates the place where the INFORMATION referring to ENTITY_CODE # CONCEPT_CODE is located. However, if such FILE_CODE does not exist, then it indicates that ENTITY_CODE # CONCEPT_CODE is in the

same DB.

ENTITY_CODE # CONCEPT_CODE: Identifier of a DB concept, where:

> ENTITY_CODE: Refers to the identification of the CODE of the entity to which INFORMATION is associated. This CODE must be provided by the entity that prepares the DB, in accordance with its classification system, to avoid ambiguities. It is recommended that this be the entity's own tax ID number.
>
> CONCEPT_CODE: Refers to a concept belonging to ENTITY_CODE and used by the DB compiling entity in its coding classification system.

When CONCEPT_CODE refers to a commercial product, such CODE shall be provided by the manufacturer and ENTITY_CODE#CONCEPT_CODE can never coincide with the designation of DB_ROOT_CODE, ENTITY_CODE or CONCEPT_CODE when it refers to a generic concept. As this commercial product has been treated as a CONCEPT, it can use all the existing registries in the format to specify its associated INFORMATION (price, graphic INFORMATION, etc.). To make use of the registries, the concept's identifier code shall be ENTITY_CODE#CONCEPT_CODE.

TECHNICAL-INFORMATION TYPE REGISTRY.

This registry includes the SPECIFICATION of other data relating to the concept, such as, for example, specific or nominal weight, physical characteristics, geometrical quantities, or physical-mechanical properties, etc.

This data may be used for other purposes, such as the calculation of thermal transmission coefficients, acoustic insulation, etc.

The Technical-INFORMATION type registry may take two forms:

If the first field is empty, it is used as a technical-INFORMATION dictionary of terms to which a description and a unit of measurement may be associated.

~X | | < IT_CODE \ IT_DESCRIPTION \ UM \ > |

If the first field identifies a concept, the INFORMATION to be specified below shall be the pair(s) of technical INFORMATION terms with their respective values.

~X | CONCEPT_CODE | < IT_CODE \ IT_VALUE \ > |

IT_CODE : Technical INFORMATION CODE described.
The following are defined:

ce	Energy cost (MJ).
eCO2	CO_2 emission (kg).
ewc	European Waste Catalogue EWC code.
m	Element mass (kg).
v	Volume (m3).

IT_DESCRIPTION: Technical-INFORMATION descriptive text, without size limitation.

UM: If the technical-INFORMATION values are numerical values, their measurement unit shall be indicated, according to the International System of Units of Measurement.

CONCEPT_CODE: CODE of the concept described, as contained in the database. It shall be unique for each ~X registry.

IT_VALUE: Alphabetical or numerical technical-INFORMATION value.

Energy-cost and CO_2-emission calculation.

When calculating the energy cost and CO_2 emission for the concepts described as discrete, ~R and ~X registries will be used. For concepts described with complex parametric descriptions in DLL, BdcNumProp and BdcPropValString will be used.

The unit energy cost of a compound element is obtained from the sum of the energy cost of the components in the price justification (which is obtained from the component's output * unit energy cost). The energy cost of single elements is a direct value.

The unit CO_2 emission of a compound element is obtained from the sum of the energy cost of the components in the price justification (which is obtained from the component's output * unit CO_2 emission). The CO_2 emission of single elements is a direct value.

For the calculation of the energy cost and CO_2 emission of a compound element, the following calculation will be applied (all decimals are used. Calculations and totals are carried out without rounding):

[]. ce = sum [C].ce
[ ].eCO2 = sum [C].eCO2
Where,
C = component
Energy cost and CO_2 emission of a price justification component [C] in a compound element:
[]. [C].ce = [C]. Ce * [ ].[C].Quantity
[ ].[C].eCO2 = [C]. eCO2 * [ ].[C].Quantity
Where []. [C].Quantity = [ ].[C].Output * [C].Factor

Example:

To describe the UM of energy cost and CO_2 emission, the following would be used:
~X| ce \ energy cost \ MJ \ eCO2 \ CO2 emissions \ kg \ |
To define the 'ce' and 'eCO2' of the single elements, the following would be used:
~X | B5221FM0 | ce \ 5.4 \ eCO2 \ 0.41 \ |

MEASUREMENT-TYPE REGISTRY.

This registry contains the measurements (quantities) in which a budget concept is involved in the breakdown of another higher-ranking concept.

In the exchange of budget files, this registry must always appear, whether there is a breakdown of measurements.

When exchanging a list of ~M registry containing a list of unstructured measurements, it is not necessary to have a root CODE or the complementary ~D registry. The operator will indicate in these cases which is the destination of the measurement.

~M | [PARENT_CODE \] CHILD_CODE | { POSITION \ } | TOTAL_MEASUREMENT | { TYPE \ COMMENT { # ID_BIM } \ UNITS \ LENGTH \ LATITUDE \ HEIGHT \ } | [LABEL] |

PARENT_CODE: CODE of the parent concept or decomposed concept of the budget.

CHILD_CODE: CODE of the child concept or decomposition line concept.

This field is optional when exchanging unstructured measurements, i.e., that do not belong to the general and complete structure of a budget.

POSITION: Position of the CHILD_CODE in the decomposition of the PARENT_CODE. This data

enables the identification of the measurement when the decomposition of the parent concept includes several child concepts with the same CODE. The numbering of positions will start at 1.

The POSITION field should always be specified in budget exchange when it is complete and structured and will indicate the complete path of the measurement described in the file structure. For example, 3 \ 5 \ 2 will indicate the measurement corresponding to chapter 3 of the file; subchapter 5 of chapter 3; and item 2 of subchapter 5. In unstructured measurements this field is optional.

TOTAL_MEASUREMENT: Must coincide with the output of the corresponding ~D-type registry. It incorporates the sum of the product of units, longitude, latitude and height or the result of expressions of each line, when reading this registry this value will be recalculated.

TYPE: Indicates the specific type of measurement line. This subfield will usually be empty. The types established in this VERSION are:

"1" Partial subtotal: This line will show the subtotal of the previous lines from the last subtotal to the line immediately preceding this one.

"2" Cumulative Subtotal: This line will show the subtotal of all previous lines from the first subtotal to the line immediately preceding this one.

"3" Expression: This indicates that an algebraic expression to be evaluated will appear in the COMMENT subfield. The operators "(",")", "+", "-", "*", "/" and "^"; the variables "a", "b", "c" and "d" (which will have as value the quantities entered in the subfields UNITS, LENGTH, LATITUDE and HEIGHT respectively); and the constant "p" for the value Pi=3.1415926 can be used. This expression will be valid until the next measurement line in which another expression is defined. Only the expression is evaluated and not multiplied by the units. The formula expressions use the criteria defined in annex 2.

COMMENT: Text in the measurement line. It may be a comment or an algebraic expression.

ID_BIM: This is optional, it arises from the need to transmit the identifier(s) of the construction elements in the BIM model to the budget programs for the bidirectional exchange of information between both platforms. These identifiers may appear in different measurement lines of other budget concepts, since the same construction element may have more than one concept associated with it. If including such identifiers, their number must match the number expressed in the field UNITS.

UNITS, LENGTH, LATITUDE, HEIGHT: Four real numbers with measurements. If any magnitude does not exist, this field will be left empty.

LABEL: This is optional and arises from the need to transmit an identifier of the concepts (chapters, subchapters, or items). This identifier is printed by several programs in the lists of a Work's measurements or budget (for example, "2.10", "A-27b", "001001", etc.), being unique for each concept (chapter, subchapter, or item) and, in general, different from the coding of the database used to prepare the budget (The "CHILD_CODE" often does not appear in the above-mentioned lists).

This registry will also be used to transmit the label of a chapter, or subchapter, of the budget. In this case, the TOTAL_MEASUREMENT field will usually be 1 and the fields TYPE, COMMENT, UNITS, LENGTH, LATITUDE and HEIGHT will not exist:
~M | [PARENT_CODE \] CHILD_CODE | { POSITION \ } | 1 | | [LABEL] |

ADD-MEASUREMENT TYPE REGISTRY.

Same as the ~M-type registry but adds the measurement lines of this registry to the existing ones instead of replacing the entire measurement as it does in the ~M-type registry.

~N | [PARENT_CODE \] CHILD_CODE | { POSITION \ } | TOTAL_MEASUREMENT | { TYPE \ COMMENT { # ID_BIM } \ UNITS \ LENGTH \ LATITUDE \ HEIGHT \ } | [LABEL] |

BIM-FILE TYPE REGISTRY.

In the ~M and ~N registries there is the subfield ID_BIM which indicates to which specific element of the BIM model each measurement line refers. All that is available is the name of the file in which the BIM model is located, which will normally be an IFC file.

The BIM file is the place in which to search for element codes and to add a list with several file names, since a construction work may consist of several linked 3D models (architecture, structure, installations, etc.).

~I | BIM_FILE.EXT {\ BIM_FILE.EXT } |

BIM_FILE.EXT: This is the name of the file containing the BIM model that includes, among other information, the identifiers of the building elements referenced in the BIM_ID field of either ~M registry or ~N registry.

KEY-TYPE REGISTRY.

This registry establishes the relationship between CODES and thesaurus descriptors, to allow concepts to be searched for by key terms.

~A | CONCEPT_CODE | < THESAURUS_KEY > |

CONCEPT_CODE: CODE of the concept described, as contained in the database.

THESAURUS_KEY: Key terms related to the concept. Compound terms (reinforced concrete, plasterboard, mixed mortar) shall be identified by means of < _ > (ASCII - 95), (reinforced concrete, plasterboard, mixed_mortar, etc.). The use of white space is not allowed.

CODE-CHANGE TYPE REGISTRY.

This registry enables the change or cancellation of the concept CODES, the sole unit of INFORMATION that could not be modified in the previously defined registries.

~B | CONCEPT_CODE | NEW_CODE |

CONCEPT_CODE: CODE of the concept to be changed or cancelled. It must exist and belong to a concept contained in the DB.

NEW_CONCEPT: NEW_CODE: New CODE for CONCEPT_CODE, it must not already exist. If this field is empty, it is understood that CONCEPT_CODE must be deleted.

ATTACHED-DOCUMENT TYPE REGISTRY.

This registry allows files with different types of information to be associated with a concept.

~F | CONCEPT_CODE | {TYPE { FILE.EXT ; } \ { [FILE_DESCRIPTION]\ } | [URL_EXT] |

Were,

CONCEPT_CODE: Concept code described in the database and its contents to which information is associated as a file.

TYPE: TYPE: Code of the type of information contained within the file.
Initially the following are defined:

0	Other.
1	Technical and manufacturing characteristics.
2	Installation, use and maintenance manual.
3	Element(s) and system(s) certificate(s).
4	Regulations and bibliography.
5	Price list.
6	Sale conditions.
7	Colour chart.
8	Scope and selection criteria.
9	Elements and systems calculation.
10	Presentation, general data, and objectives of the company.
11	Company certificate/s.
12	Works completed.
13	Image.

FILE.EXT: Name of the file with extension containing CODE_CONCEPT information.
A (main) file can contain linked files, separated with the <; > (ASCII-59) character, with the main file being the first to be displayed. All files (including linked ones) can be in the same directory in which the. BC3 file(s) including their reference are located or in the URL specified.
In addition to the extensions allowed in the ~G, ~L and ~J registries, the following extensions are added using standardized programs of general use for reference, to check and verify the content of the file:

.PDF extension	Adobe Acrobat Reader.
.AVI extensión	VLC Media Player.
.PPT extension	Microsoft Power Point.

FILE_DESCRIPTION: Brief description of the information contained in the file.

URL_EXT: is an optional field. If it is not empty, it is the URL to be added to the URL_BASE to find the document. URL_BASE is defined in the ~V registry. It behaves as follows: first the graphic is searched for in the local directory and if it is not there, it is searched for in URL_BASE + URL_EXT + FILE.EXT.

FIEBDC-3 FORMAT. SUMMARY.

~V | [FILE_OWNERSHIP] | [FORMAT_VERSION] [\ DDMMYYYY] | [EMISSION_PROGRAM] | [HEADER] \ { IDENTIFICATION_LABEL \ } | [CHARACTER_SET] | [COMMENT] | [INFORMATION TYPE] | [CERTIFICATION NUMBER] | [CERTIFICATION DATE] | [URL_BASE] |

~K | { DN \ DD \ DS \ DR \ DI \ DP \ DC \ DM \ CURRENCY \ } | CI \ GG \ BI \ REDUCTION \ VAT | { DRC \ DC \ \ DFS \ DRS \ \ DUO \ DI \ DES \ DN \ DD \ DS \ DSP \ DEC \ CURRENCY \ } | [n] |

~C | CODE { \ CODE } | [UNIT] | [SUMMARY] | { PRICE \ } | { DATE \ } | [TYPE] |

~D | PARENT_CODE | < CHILD_CODE \ [FACTOR] \ [OUTPUT] \ > | < CHILD_CODE \ [FACTOR] \ [OUTPUT] \ { PERCENTAGE_CODE ; } \ > |

~Y | PARENT_CODE | < CHILD_CODE \ [FACTOR] \ [OUTPUT] \ > | < CHILD_CODE \ [FACTOR] \ [OUTPUT] \ { PERCENTAGE_CODE ; } \ >|

~ R | PARENT_CODE | { DECOMPOSITION_TYPE \ CHILD_CODE \ { PROPERTY \ VALUE \ [UM] \ } | } |

~T | CONCEPT_CODE | DESCRIPTIVE_TEXT |

~P | [PARAMETRIC_DESCRIPTION] | [DLL_NAME] |

~P | FAMILY_CODE | [PARAMETRIC_DESCRIPTION] |

~L | | < SPECIFICATION_SECTION_CODE \ [SPECIFICATION_SECTION_LABEL] \ > |

~L | CONCEPT_CODE | { SPECIFICATION_SECTION_CODE \ SPECIFICATION_SECTION_TEXT \ } | { SPECIFICATION_SECTION_CODE \ TEXT_FILE_RTF \ } | { SPECIFICATION_SECTION_CODE \ TEXT_FILE_HTM \ } |

~Q | < CONCEPT_CODE \ > | {SPECIFICATION_SECTION_CODE \ PARAGRAPH_CODE \ { SCOPE_ABREV ; } \ } |

~J | PARAGRAPH_CODE | [PARAGRAPH_TEXT] | | [PARAGRAPH_FILE_RTF] | [PARAGRAPH_FILE_HTM] |

~W | < SCOPE_ABREV \ [SCOPE] \ > |

~G | CONCEPT_CODE | < GRAPH_FILE.EXT \ > | [URL_EXT] |

~E | ENTITY_CODE | [SUMMARY] | [NAME] | { [TYPE] \ [SUBNAME] \ [ADDRESS] \ [PC] \ [TOWN] \ [PROVINCE] \ [COUNTRY] \ { TELEPHONE ; } \ { FAX ; } \ { CONTACT_PERSON ; } \ } | [TAX ID] \ [WEB] \ [EMAIL] \ |

~O | DB_ROOT_CODE # CONCEPT_CODE | | < FILE_CODE \ ENTITY_CODE # CONCEPT_CODE \ > |

~X | | < IT_CODE \ IT_DESCRIPTION \ UM \ > |

~X | CONCEPT_CODE | < IT_CODE \ IT_VALUE \ > |

~M | [PARENT_CODE \ ] CHILD_CODE | { POSITION \ } | TOTAL_MEASUREMENT | { TYPE \ COMMENT { # ID_BIM } \ UNITS \ LENGTH \ LATITUDE \ HEIGHT \ } | [LABEL] |

~N | [PARENT_CODE \ ] CHILD_CODE | { POSITION \ } | TOTAL_MEASUREMENT | { TYPE \ COMMENT { # ID_BIM } \ UNITS \ LENGTH \ LATITUDE \ HEIGHT \ } | [LABEL] |

~I | FILE.EXT { \ FILE.EXT } |

~A | CONCEPT_CODE | < THESAURUS_KEY > |

~B | CONCEPT_CODE | NEW_CODE |

~F | CONCEPT_CODE| { TYPE \ { FILE.EXT ; } \ [FILE_DESCRIPTION] \ } | [URL_EXT] |

Annex 1. Changes with respect to previous versions: 3/2016, 3/2012, 3/2007, 3/2004, 3/2002, 3/98 and 3/95.

Summary of changes in version 3/2020 with respect to 3/2016:

- Specification: Database information is now only given in a single file. Furthermore, it is specified how several certifications can be collected at the same time in a single file.
- New BIM-FILE type ~I registry as a complement to ~M and ~N.
- Annex 6. Currencies. According to ISO 4217.
- New Annex 9 on Criteria for the assignment of references in IFC to price banks in FIEBDC format.

Changes to the API standard for compiled parametric descriptions:

- Introduction: Recommendation for both budget programs and databases compiled into DLLs to create 64-bit versions.
- Example: The example has been changed and is now ready to be compiled with Microsoft Visual Studio 2010 multibyte character set as a 32-bit and 64-bit DLL.

Summary of changes in version 3/2016 with respect to 3/2012:

- Specification: Specifies how to transfer information from databases on the Internet to applications on local computers using FIEBDC.
- ~K COEFFICIENT-type registry. It clarifies what to do when receiving quantities with more decimal digits than those specified in the ~K registry of a FIEBDC file and, on the other hand, what to do in the case of receiving a FIEBDC file without the ~K registry.
- ~M MEASUREMENTS-type registry and ~N ADD-MEASUREMENTS type registry: Specifies how to transmit the identifier(s) of the construction elements in the BIM model to the estimating programs for the bidirectional exchange of information between both platforms.
- Annex 2. Modification of the parametric concepts in the parameter-dependent coding model, whether it has a DLL, to have more than four parameters.
- Annex 4: The types of concepts obtained from the indexes and polynomial formulas for BOE price revisions are split between those of RD 1359/2011 and those prior to said RD.
- Annex 7. Update of the regulations governing measurement units.
- Annex 8. Update of the definition of the BME and CEB budget types, as well as adding the work-unit calculation system, in accordance with Real Decreto 1098/2001 Reglamento General de la Ley de Contratos de las Administraciones Públicas.

Summary of changes in version 3/2012 with respect to 3/2007:

- ~V VERSION-type registry: New URL_BASE field to provide documents and graphics associated to concepts in a url.
- ~C CONCEPT-type registry. Extension of concept TYPE “4” and “5”.
- ~D DECOMPOSITION-type registry: new field 3
- ~Y ADD-DECOMPOSITION type registry. New field 3
- New ~R WASTE-DECOMPOSITION type registry.
- ~G GRAPHIC-INFORMATION type registry and ~F ATTACHED-DOCUMENT type registry. New URL_EXT field to add to the URL_BASE IN ORDER TO supply documents and graphics associated to concepts. In the ~F registry the “13” TYPE document is added.
- ~O COMMERCIAL-INFORMATION type registry. Clarification of the CODE_ENTITY_CODE # CODE_CONCEPT field.
- ~X TECHNICAL-INFORMATION type registry. The types “ce”, “eCO2”, “ewc”, “m” and “v” of CODE_IT are specified, and the energy-cost and CO2-emission calculation system is attached.
- Annex 4: Concept types 4 and 5 are added.

Changes to the API standard for compiled parametric descriptions:

- New BdcUsos function.
- New BdcDesCodigoPorcentaje function.

- BdcDocCodigo function. Added that files can contain an URL_EXT.
- New BdcNumProp funciton.
- New BdcPropValString function.
- New BdcNumComponentes function.
- New BdcCodigoComponente function.
- New BdcNumPropComponente function.
- New BdcComponenteValString function.

Summary of changes in version 3/2007 with respect to 3/2004:

- Specification: Clarification on the prohibition of using blank spaces in concept codes: "including A-Z, a-z, 0-9, ñ, Ñ," [...] "Excluding any other character such as space, tab, etc".
- ~V VERSION-type registry: New INFORMATION TYPE, CERTIFICATION NUMBER and CERTIFICATION DATE fields.
- ~K COEFFICIENT-type registry. Fields DRO and DFO are deleted. DSP and DEC fields are added.
- ~C CONCEPT-type registry. Code limited to 20 characters. Maximum recommendation of 64 characters for the summary.
- ~D DECOMPOSITION-type registry: Remove factor and output from chapters.
- ~M MEASUREMENT-type registry: Clarification on the use of expressions.
- Annex 4: Clarification that the Unit-Budget type should not use the # of the chapter type.

Summary of changes in version 3/2004 with respect to 3/2002:

- Specification: <a> notation is included for optional fields.
- ~V VERSION -type registry: New COMMENT field.
- ~K COEFFICIENT-type registry. Default decimals are set, with modifications of some decimal fields.
- ~D DECOMPOSITION-type registry: Sets the default value for output.
- ~C CONCEPT-type registry. Modifications of the CODE and PRICE fields.
- ~M and ~N MEASUREMENT and ADD-MEASUREMENT type registry: New LABEL field.
- Registries updated with notations to define mandatory and optional fields: ~V, ~D, ~Y, ~T, ~L, ~Q, ~W, ~G, ~O, ~X, ~A and ~B.
- New ATTACHED-DOCUMENT type ~F registry.

Changes to the API standard for compiled parametric descriptions:
- New function BdcTipoDescripcion().
- New function BdcInvalidos().
- Clarification of the BdcPliego() function.
- Modification of the parameters of the BdcComercCodigo() function.
- New error message BDCERR_PARAMETER_INCORRECT.
- New functions BdcDocNumero() and BdcDocCodigo() related to the attached-document type ~F registry.

Classification in types of concepts:
- New Unit-Budget type.

The list of sections and registries of FIEBDC-3/2016 affected by extensions and/or modifications is shown below:

Section	**3/2016**	**3/2012**	**3/2007**	**3/2004**	**3/2002**	**3/98**	**3/95**
PRESENTATION.		X	X	X	X	X	X
FIEBDC-3 FORMAT. SPECIFICATION.	X	X		X	X	X	X
~V OWNERSHIP AND VERSION-type registry.			X	X	X	X	X
~K. COEFFICIENT-type registry.		X		X	X	X	X
~C. CONCEPT-type			X	X	X	X	X

registry.							
~D. DECOMPOSITION-type registry.			X	X	X	X	X
~Y. ADD-DECOMPOSITION type registry.			X		X	X	X
~R. WASTE-DECOMPOSITION type registry.			X				
~T. TEXT-type registry.					X	X	X
~P. PARAMETRIC-DESCRIPTION type registry.							X
~L. SPECIFICATIONS-type registry.					X	X	X
~Q. SPECIFICATIONS-type registry.					X	X	X
~W. GEOGRAPHIC-SCOPE type registry.					X	X	X
~G. TECHNICAL-INFORMATION type registry.			X		X	X	X
~E. ENTITY-type registry.						X	X
~O. COMMERCIAL-RELATIONSHIP type registry.			X		X	X	X
~X. TECHNICAL-INFORMATION type registry.			X		X	X	X
~M. MEASUREMENT-type registry.		X		X	X	X	X
~N. ADD-MEASUREMENT type registry.		X			X	X	X
~I. BIM-FILE type registry.	X						
~A. KEY-type registry.					X	X	X
~B. CODE-CHANGE type registry.					X	X	X
~F. ATTACHED-DOCUMENT type registry.			X		X	X	X
Annex 2. Parametric description: STANDARD format.		X					
Annex 3. Parametric description: API STANDARD for parametric descriptions compiled in DLL.	X		X		X	X	X
Annex 4. Classification in types of Concepts:		X	X	X	X	X	X
Annex 6. Currencies	X						
Annex 7. Measurement units.		X					
Annex 8. Definitions of different types of Budgets.		X					
Annex 9. Criteria for the assignment of references in IFC to price banks in FIEBDC format.	X						

Annex 2. PARAMETRIC DESCRIPTION: STANDARD format.

A parametric concept is a concept that defines its CODE, summary, text, specifications, decomposition and business INFORMATION in a parametric way, i.e., in a variable way by means of tables and arithmetic and logical expressions as a function of parameters.

The parametric description contains the following instructions:

Variables are defined:
%A %B %C %D %F %G %H %I %J %K: Selected parameters of "a" to "z" concept ~ 1 to 26.
%O %P %Q %R %S %T %U %V %W %X: Selected parameters of "a" to "z" concept ~ 1 to 26.

%E:	Variable defining the error conditions.
$A $B $C $D $F $G $H $I $J $K:	Selected parameter texts of the concept.
$O $P $Q $R $S $T $U $V $W $X:	Selected parameter texts of the work.
$E:	Variable defining the error texts.

Similarly, variables %O to %X and $O to $X would take the value corresponding to the work's general parameter values.

Any variable from "A" to "Z", either numeric (%) or alphanumeric ($), can be defined or redefined by any number of dimensions for later use in expressions.

The constants "a" to "z" are defined with numeric values from 1 to 26 respectively, to allow mnemonic referencing of the parameters. For the use of other types of characters, the substitution character to be used shall be determined in the text of the selected parameter option by prefixing it with a special character "!". If such a character does not exist, the substitution is made by relating the character to the position it occupies.

Example: PBPO.2$ M3 Concrete $B $A
\ CONSISTENCY \ plastic \ fluid \ soft \
\ RESISTANCE \ H-125 \ H-150 \ H-175 \ H-200 \ H-200 \
The derivative PBPO.2aa would be: M3 Concrete H-125 plastic

With special character:
\ CONSISTENCY \! p plastic \! f fluid \ !b soft \
\ RESISTANCE \ !2 H-125 \ !5 H-150 \ !7 H-175 \ !0 H-200 \
The derivative would be: PBPO.2p2 M3 Concrete H-125 plastic

Numeric variables must allow double-precision (64-bit) floating-point real values and alphanumeric variables must be able to store text of any size.

Any variable can be defined, with the same assignment, with any number and size of dimensions (up to 4). All dimensions must be made explicit in the DEFINITION of dimensions.

```
%U =.....       # defines a variable with a numerical data.
$X (8) =.....   # defines a list of 8 alphanumerical data.
%V (3,4) = ..... # defines a table with 3 rows and 4 columns of n data.
```

The variables %E and $E are special variables for returning errors caused by inconsistent parameter selections. In a sequential evaluation of expressions, if a variable %E takes a value other than 0 in an expression, there has been an error, the evaluation of expressions is interrupted and the content of the $E variable is presented where the text of the error produced is stored.
There may be multiple assignments of %E, each of them preceded by its corresponding error text, the $E assignment.

Alphanumeric constants will be defined in quotation marks ($I="including proportional part").

In the parametric description, the following types of statements will be found:

PARAMETER-LABEL STATEMENT:

\ { <Parameter LABEL> { <Option LABEL> }

The defined parameters, up to 10, will be assigned to the ABCDFGHIJK variables in the order in which they are found.

NUMERICAL-ASSIGNMENT STATEMENT:

<numeric variable> = <numeric expression>

ALPHANUMERIC-ASSIGNMENT STATEMENT:

<alphanumeric variable> = <alphanumeric expression>.

OUTPUT STATEMENT (DECOMPOSED CONCEPTS):

<CODE substitution text> : <numeric expression> [: <exp.num.>] One or, optionally, two outputs can be defined, the default of the optional output is 1.

AUXILIARY-EQUIPMENT STATEMENT:

%: <numeric expression> (percentage).

%%: <numeric expression> (as a percentage of one)

PRICE STATEMENT (SIMPLE CONCEPTS): <numerical expression>.

If a set of output statements appear together, as a decomposition and a price statement, the price statement will take precedence, ignoring the output statements.

COMMENT STATEMENT:

\ COMMENT \ or \ C \ <COMMENT text> \

If comment text exists, it will be presented as a parameter-selection aid next to the parameter options.

SUBSTITUTION STATEMENT:

\ SUMMARY \ or \ R \ <substitution text of the summary text>.

\ TEXT \ or \ T \ { <substitution text for descriptive text>.

\ SPECIFICATIONS \ or \ P \ { <substitution text for specifications> \ }

\ KEY \ or \ K \ { <substitution text for key> \ }

\ COMMERCIAL \ or \ F \ { <substitution text for CODE>\ <numerical_expression>\}

A statement is considered continuous on the next line if:

- It ends in an operator
- It ends without closing quotation marks ""
- It begins with “\” and does not end with “\”.

<constants> PI, numbers, "text" ...

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

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

<numeric expression>:

Those resulting in a number as a function of numerical constants and variables, logical expressions, functions, and operators.

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

<alphanumeric expression>:

These result in text as a function of alphanumeric constants and variables, operators, and numeric functions.

An alphanumeric expression can include logical expressions.

for example: $I="proportional part "+" of losses "*(%A>a)

add "of losses" to $I if the current value of %A is greater than <a> or 1.

<logical expressions>:
These are those that result in TRUE or FALSE. In numeric expressions true is considered as 1 and false as 0, in alphanumeric false is indicated as the deletion of text.
%I = 323*(%A=a) + 345*(%A=b) + 1523*($I=$A & $J=$B)
$I = "white "*(%C=c) + "black "*(%C=d)

<substitution text>:
In substitution texts the INFORMATION is a constant text (without quotes) with variables embedded within it. The characters $ and % immediately followed by a letter from A to Z are considered variables.
In substitution texts, alphanumeric variables are replaced by their corresponding text contents, while numeric variables are replaced by the constants from "a" to "z" corresponding to the numeric value of their contents.
In the output expression, the first part of the sentence is a substitution text that, once the variables have been substituted, will be the CODE of the concept to which the numerical expression of the second part of the expression corresponds as output. If the result is 0, the statement is ignored, and that component or decomposition line is not considered.

NOTATIONAL CONVENTIONS (EBNF):

[a]	Indicates nothing or "a".
{a}	Indicates zero or more "a" occurrence.
[a-b]	Indicates any value from "a" to "b", both inclusive.
[abc]	Indicates any of the values "a", "b" or "c"
<abc>	Indicates informal description
abc	Indicates terminal symbol

%[A-Z]	Numeric variable
$[A-Z]	Alphanumeric variable

Predefined variables:

[%$][ABCDFGHIJKLM]	Concept parameters.
[%$][OPQRSTUVWX]	Work parameters
[%$]E	Special variable for error reporting
[%$][A-Z][(dim{,dim})]	Definable variables
#	Comments (text between this character and the following end of line, exclusive, is not considered)
,	Data separator
:	Output DEFINITION
::	Price DEFINITION
%:	DEFINITION of auxiliary equipment in percent
%%:	DEFINITION of auxiliary equipment expressed per unit

NUMERICAL OPERATORS (From lowest to highest precedence):

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Operator raised to

LOGICAL OPERATORS (From lowest to highest precedence):

@	Logical operator OR
&	Logical operator AND
<	Least
>	Greater
<=	Less than or equal to
>=	Greater than or equal to

=	Equal
<>	Different
!	Logical operator NO

FUNCTIONS -- RETURN VALUE:

ABS(n)	Absolute value of "n".
INT(n)	Integer part of "n".
ROUND(n,d)	Rounding of "n" to "d" decimals
SIN(n)	Sine (sexagesimal degrees)
COS(n)	Cosine (sexagesimal degrees)
TAN(n)	Tangent (sexagesimal degrees)
ASIN(n)	Arc sine (sg)
ACOS(n)	Arc cosine (sg)
ATAN(n)	Arc tangent (sg)
ATAN2(x,y)	Arc tangent with two parameters "x" and "y".
SQRT(n)	Square root of "n".
ATOF(a)	Conversion from alphanumeric "a" to numeric
FTOA(n)	Conversion from numeric "n" to alphanumeric

Each instruction will be on a separate line, unless the instruction ends in an operator, in which case it is considered to continue the next line.
If a line ends without closing the quotation marks "" or delimiter '\', it is considered to continue on the next line. The end-of-line characters (ASCII-13 + ASCII-10) contained in the parametric descriptions will be retained when reformatting.

CONTROL OF SELECTION ERRORS.

It is common to find many possible parameter combinations but have few of them resolved. To prevent the system operator from trying different parameter combinations and returning an error message in all of them, the system must be able to guide the operator to select the proper combinations.
Each time the operator defines a parameter, the system shall evaluate all possible statements. For statements of the type: %E= [...], parameter function.

If all parameters involved in the expression are known, the expression shall be evaluated and if the result is an ERROR, the previous DEFINITION of $E shall be presented with the error message.

If all parameters are known except one, the unknown parameter will be given values and the expression will be evaluated until all valid values of the parameter are traversed. In one way or another, the system “will mark” the ERROR-producing values of the parameter in question on the selection screen, to help the operator to select the correct combinations.

Whenever a parameter is defined or redefined, the system will update all values marked in the screen. For example, setting the LABELs for options whose selection will not be compatible with the previously selected parameters to “half brightness”.

This selection-error checking system can be easily implemented on any device but requires the publishers of the parametric decompositions to explicitly define the incorrect parameter combinations, as this method will not find non-permitted combinations when the parametric decomposition calls other decompositions or parametric prices.

PROCEDURE FOR READING PARAMETRIC DESCRIPTIONS.

Run the parametric description by carrying out the following steps:

1. Remove everything from "#" inclusive, up to the following line change.

2. Change tabs (9) to " " characters (32)

3. Remove " " characters (32) before and after the "\" characters.

4. Combine lines, removing the end of the line, into lines that begin with "\" rather than ending with "\", ending with an operator and in the data separation of a matrix variable.

5. Remove all " " characters (32) in unquoted areas ("...") or non-delimited areas ("\...\").

6. Remove empty lines.

7. Read and sequentially evaluate sentences as follows:

   If the sentence starts with "\", read the LABEL up to the following "\", if the LABEL is:
   COMMENT or C- Reserved word or character that identifies the following LABEL between "\" as a comment to the parameter selection.
   SUMMARY or R- Reserved word or character that identifies the following LABEL between "\" as the substitution text for the concept's summary.
   TEXT or T- Reserved word or character that identifies the following LABEL between "\" as the substitution text for the concept's descriptive text.
   SPECIFICATION or P- Reserved word or character that identifies the following LABELs between "\" as the substitution texts for the different sections of the specification.
   KEYs or K- Reserved word or character that identifies the following LABELs between "\" as the substitution texts for the key words associated with the concept.
   COMMERCIAL or F- Reserved word or character that identifies the following LABELs between "\" as the substitution texts and for the concept's commercial INFORMATION.
   Any other LABEL will identify the name of the following parameter and the following LABELS between "\" as the LABELs for the options of said parameter.
   If the sentence starts with "::"", the rest of it must be a numeric expression indicating the Price – only in families of simple concepts (no decomposition) and there can only be one sentence of this type.

   If the sentence begins with "%", the rest of it must be a numeric expression indicating the Percentage of Auxiliary Equipment – there can only be one sentence of this type.

   Otherwise, if the sentence contains the character ":" the section before this is the CODE's substitution text for a line of decomposition, and that which follows is a numeric expression (or two separated by ":"), indicating the output or outputs of said line of decomposition.

   In those cases where a "%" character may appear followed by an alphabetic character, which is considered as such and not as a substitution variable, this should use "%%" to avoid the ambiguity that may arise between a numeric variable to be substituted, an auxiliary-equipment sentence, or a text.

   The rest of the sentences must be assignment statements of the form variable/s = expression/s

SUMMARY OF SENTENCE TYPES.

After carrying out the filter described above, each line or character strip ending in (ASCII-13)(ASCII-10), will be one of the following types of sentences:

```
{ \ parameter_LABEL \ { parameter_option \ } (13)(10) }
{ variable = expression (13)(10) }
{ CODE : output [ : output ] (13)(10) }
[ %: ó %%: auxiliary_equipment (13)(10) ]  % (percent)  %% (expressed per unit)
[ :: price_expression (13)(10) ]
[ \ COMMENT \ or \ C \ comment_text \ (13)(10) ]
[ \ SUMMARY \ or \ R \ summary_text \ (13)(10) ]
```

[\ TEXT \ or \ T \ descriptive_text \ (13)(10)]
[\ STATEMENT \ or \ P \ { statement_facet_text \ }(13)(10)]
[\ KEYS \ or \ K \ { key_term \ }(13)(10)]
[\ COMMERCIAL \ or \ F \ { commercial_product_CODE \ rate \ } (13)(10)]

Annex 3. PARAMETRIC DESCRIPTION: STANDARD API for parametric descriptions compiled in DLL

INTRODUCTION.

Given the need presented by the developers of the parametric databases to broaden the possibilities of parametric-description language, to be able to compile this for efficiency and data-protection purposes and to enable copy protection for parametric databases, the following SPECIFICATION is established.

In this document, the necessary components for the development of parametric descriptions in any language for Windows applications (C, C++, Pascal, Fortran, etc.) is defined, without limitation. The DEFINITION of a STANDARD API in C is included, along with an example 32-bit database in DLL format developed in C++ and an example application with the implementation of the interface, with the API in C, both defined in Microsoft Visual C++. The interface with the API can be implemented for other compilers and languages to access the same DLLs.

In other words, it is possible to construct a database compliant with this API, using any programming language that allows the development of Windows dynamic-link libraries (DLL). Likewise, it is possible to build a program that reads any database with these characteristics, using Windows application languages.

The set of characters used in the texts returned by the API functions will be that specified in the registry ~V.

A budget file with the extension .BC3 should not deliver information to the DLL for parameters.

We recommend that budget-program developers and publishers of databases with parametric descriptions compiled in DLLs add 64-bit versions and that they maintain the 32-bit versions until the 64-bit versions are in full use, at which time they should stop using the said 32-bit versions.

FILES TO BE CONTAINED WITHIN A DATABASE.

A database intended to distribute parametric definitions compiled in DLL must contain the following files:

base.bc3 ASCII database file or files in FIEBDC-3/2020 format. The ~P registries of the concepts whose parametric description is accessed through the file “base.dll” will have the PARAMETRIC_DESCRIPTION field empty. The concept CODE for this registry must match the CODE of the corresponding ~C registry and the CODE used in the calls to the API function, including any possible pound signs (“#”). Example: ~P|ABCD12$| |
The ~P registry corresponding to the global parametric will have an empty PARAMETRIC_DESCRIPTION field, and a third field with the name of the DLL file in which the database API’s features appear. Example: ~P| | | BASE.DLL |

base.dll In this file – unique for each database and which can have any name, if it has the extension .DLL – we find the API functions that the database offers to applications for these to obtain the INFORMATION contained therein.

The parametric DEFINITION of the concepts implemented in this way can be found in the same file as the API functions (the “base.dll” file) or located in any other file, as the database developer wishes. Applications can only access the API functions included in the “base.dll” file, and these will be responsible for accessing the INFORMATION in the form implemented by the database developer.

DEFINITION OF THE API: FIEBDC.H

Single file defining the STANDARD. In this file, the API is defined in C, which the parametric descriptions in DLL offer to the applications. We recommend including all API functions in the Databases, even if it does not use them all. This interface allows parametric definitions for an unlimited number of parameters and unlimited options per parameter. Two models of codifications are supported:

1. A codification model independent of parameters, in which the CODE for a parametric concept is completely free and the number of characters in the CODE is independent of the number of parameters. In this model, the codes of the parametric families do not need to include the character "$".

2. A model dependent on parameters. This is the model defined by FIEBDC-3/95 and in which the CODE for a parametric concept must include the symbol "$" in seventh position, and in which the options 0 to 25 of each parameter are assigned from "a" to "z", with the number of options per parameter in this VERSION being extended with ranges "A" to "Z" and "0" to "9", bringing this number to 62 (from 0 to 61).

In order for the programs to determine whether a database responds to one model or another, the function BdcCodificacion() has been defined, as specified below and which indicates whether the codification system used in the database is dependent or independent.

If the first model is adopted, it will not be possible to deduce from a "ABCDEFGHIJ" concept CODE whether this is a parametric derivative, nor from which parametric concept it is derived nor with which of its parameters' values. As a result, the following search criteria is established:

1. If the concept exists with this CODE in the database, it will choose said concept.

2. If not, it will try to localize it within the database as belonging to a FIEBDC-3/95-style parametric concept. In the example, it will try to search for the parametric concept "ABCDEF$" and will pass the parameters "GHIJ" to it (which implies passing it four parameter values "31", "32", "33" and "34" respectively).

3. If these do not exist, it will try to localize this within the DLL. If this has a dependent codification model, it will use the same criteria as in the previous point: in the example, it will search for the parametric concept "ABCDEF$" and will pass it the parameters "GHIJ". If the database has an independent model, it will use the function "BdcDecodifica()", as specified below.

4. If none of the previous conditions apply, it is assumed that the concept does not exist in the database.

See the definition of the API in the attached file "fiebdc.h".

SPECIFICATIONS OF THE API FUNCTIONS.

1. GENERAL FUNCTIONS.

```
LONG EXPORT BdcCodificacion (
        VOID
);
```

Aim
Indicates whether the database will use a codification model which is dependent or independent of the number and value of the parameters.

Value returned
This will return “0” if the codification follows a dependent model (in the FIEBDC-3/95 style) and “1” if it follows an independent model.

```
LONG EXPORT BdcTipoPliego (
        VOID
);
```

Aim
Indicates what type or types of specifications of conditions are implemented in the database. Said models are specified in the “SPECIFICATION-TYPE REGISTRIES” section of the format specifications.

Value returned
This will return “0” if no type of specification has been implemented.
This will return “1” if model one of the specification texts has been implemented. In this case, the function BdcPliego() will be used to obtain the specification texts.
This will return “2” if model two of the specification texts has been implemented. In this case, the functions BdcCodigoParrafo() and BdcTextoParrafo() will be used to obtain the specification texts.
It will return “3” if both models one and two have been implemented.

```
LPCSTR EXPORT BdcFecha (
        VOID
);
```

Aim
Obtains the Database date.

Value returned
Returns the Database date in format DDMMYYYY, where DD represents the day with two digits, MM the month and AAAA the year.
The function itself is responsible for allocating memory to the pointer. In case of error, the function will return “NULL”. To obtain more INFORMATION about the error produced, call the function BdcError().

2. FUNCTIONS REFERRING TO THE GLOBAL PARAMETRIC.

2.1. Accessible at any time.

2.1.1. Obtaining its parameters.

```
LONG EXPORT BdcGloParNumero (
        VOID
);
```

Aim
Obtains the number of parameters for the global parametric concept.

Value returned
Returns the number of parameters. In case of error, the function will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcGloError().

```
LONG EXPORT BdcGloOpcNumero (
        LONG par        // number of the concept’s parameter
);
```

Aim
Obtains the number of options of which the ‘par’ parameter consists of.

Parameters
par: Number of the parameter. This should be a value between “0” and “n-1”, with “n” being the number of parameters of the global parametric concept.

Value returned
This returns the number of options for the “par” parameter. If an error is produced, it will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcGloError().

```
LPCSTR EXPORT BdcGloParRotulo (
        LONG par        // number of the concept’s parameter
);
```

Aim
Obtains the label that identifies the “par” parameter for the concept.

Parameters
par: Number of the parameter. This should be a value between “0” and “n-1”, with “n” being the number of parameters of the global parametric concept.

Value returned
Returns the label that identifies the “par” parameter for the concept, as a constant far pointer to a string. The function itself is responsible for allocating memory to the pointer. In case of error, the function will return “NULL”. To obtain more INFORMATION about the error produced, call the function BdcGloError().

```
LPCSTR EXPORT BdcGloOpcRotulo (
        LONG par,       // number of the concept’s parameter
        LONG opt        // number of the parameter’s option
);
```

Aim

Obtains the label that identifies the “opt” option of the concept’s “par” parameter.

Parameters
par: Number of the parameter. This should be a value between “0” and “n-1”, with “n” being the number of parameters of the global parametric concept.
opt: Number of the parameter. This should be a value between “0” and “n-1”, with “n” being the number of options within the global parametric concept’s “par” parameter.

Value returned
Returns the label that identifies the “opt” option of the concept’s “par” parameter, as a constant far pointer to a string. The function itself is responsible for allocating memory to the pointer. In case of error, the function will return “NULL”. To obtain more INFORMATION about the error produced, call the function BdcGloError().

2.1.2. Messages / Error CODES.

```
LONG EXPORT BdcGloError (
        LPCSTR *err     // error message returned
);
```

Aim
Obtains the type of error produced. Once read, the error CODE will launch.

Parameters
err: Pointer to a constant far pointer to a string. The error message referring to the error produced is stored therein. The function is responsible for allocating memory to the pointer. If no message has been defined for the error that has been produced, “*err” shall point to the empty string “”.

Value returned
Returns the error CODE produced. See the end of the “error message CODES” section for more INFORMATION.

2.1.3. Allocating options to the parameters.

```
BOOL EXPORT BdcGloCalcula (
        LPLONG optl,    // list of options for the parameters
);
```

Aim
Assigns the values of the global parametric concept’s parameters.

Parameters
optl: Pointer to a vector (array) of LONGs with the options desired for each parameter. The options are numbered, starting from zero.

Value returned
Returns “0” if executed correctly. In case of error, this will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcGloError().

3. FUNCTIONS REFERRING TO ALL OTHER PARAMETRICS.

3.1. Accessible at any time.

3.1.1. Reading a parametric concept.

```
HANDLE EXPORT BdcLee (
        LPCSTR cod     // concept CODE
);
```

Aim
Reads the parametric code identified by 'cod'.

Parameters
cod: Constant far pointer to a string with the parametric concept CODE to be read. If a dependent codification model is used, it is assumed that said CODE has 7 characters and that the seventh is "$". Any characters may appear within the CODE, except for 0x00 (which indicates the end of the CODE).

Value returned
If the function finds the parametric, it will return a HANDLE other than zero. In case of error, or if the parametric does not exist, the function will return zero.

3.1.2. Reading a parametric concept from the derivative's complete CODE.

```
HANDLE Export BdcDecodifica (
        LPCSTR cod,    // Complete CODE for the parametric derivative
        LPLONG optl    // pointer to the memory space to be filled by the options
);
```

Aim
Reads the parametric concept to which the 'cod' CODE concept belongs. The HANDLE and the 'optl' options returned can be directly used in a call to BdcCalcula().

Parameters
cod: Constant far pointer to a string with the parametric concept CODE for which the parametric concept to which it belongs is to be obtained. Any characters may appear within the CODE, except for 0x00 (which indicates the end of the CODE).
optl: Pointer to a vector (array) of LONGs in which the function will return the options to which the parametric derivative corresponds. The array should be previously dimensioned with at least the number of the concept's parameters. The options are numbered, starting from zero.

Value returned
If the function finds the parametric, it will return a HANDLE other than zero. In case of error, or if no parametric concept exists for which the "cod" concept is derived, the function will return zero.

```
LPCSTR EXPORT BdcFamilia (
        LPCSTR cod)    // Complete CODE for the parametric derivative
);
```

Aim
Reads the code for the parametric concept to which the 'cod' CODE concept belongs.

Parameters

cod: Constant far pointer to a string with the parametric concept CODE for which the parametric concept to which it belongs is to be obtained. Any characters may appear within the CODE, except for 0x00 (which indicates the end of the CODE).

Value returned
If the function finds the parametric, it will return its code as a constant far pointer to a string of characters. The function itself is responsible for allocating memory to the pointer. In case of error, or if no parametric concept exists for which the "cod" concept is derived, the function will return "NULL". To obtain more INFORMATION about the error produced, call the function BdcError().

3.1.3. Searching for concepts based on a text or code

```
BOOL EXPORT BdcUsos (
LPCSTR text, // code or text to be searched
LONG where // place in which the text or code will be searched for
LPCSTR *lcode // list of codes where the text/code is being used
);
```

Aim
Obtains a list of codes in which a concept's text or code is used.

Parameters
Text: Indicates the text or code to be searched for.
Where: Parameter that indicates to the function whether the text is a code or a text (which may be a string). The possible values are 1: when the parameter is a code, or 2: when the parameter is a text.
lcode: Text in plain-text format containing the list of codes in which the text or code is used, separated by the ASCII 124 field separator.

Value returned
Returns "0" if executed correctly. In case of error, the function will return "-1". To obtain more INFORMATION about the error produced, call the function BdcError().

If the parameter "where" is 1, the text corresponds to a bank code and the function will return the list of concepts in which the code appears directly in the justification, or in one of its components. If the parameter "where" is 2, the first parameter corresponds to a text and the function will return the list of concepts that contain the text in the definition (summary and complete).

3.1.4. Messages / Error CODES.

```
LONG EXPORT BdcError (
        HANDLE h,       // concept identifier
        LPCSTR *err     // error message returned
);
```

Aim
Obtains the type of error produced.

Parameters
h: Concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.
err: Pointer to a constant far pointer to a string. The error message referring to the error produced is stored therein. The function is responsible for allocating memory to the pointer. If no message has been defined for the error that has been produced, "*err" shall point to the empty string "".

Value returned
Returns the error CODE produced. See the end of the “error message CODES” section for more INFORMATION.

3.2. Accessible after BdcLee.

3.2.1. Obtaining its parameters.

```
LONG EXPORT BdcParNumero (
        HANDLE h        // concept identifier
);
```

Aim
Obtains the number of parameters for the parametric concept.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.

Value returned
Returns the number of parameters. In case of error, the function will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcError().

```
LONG EXPORT BdcOpcNumero (
        HANDLE h,       // concept identifier
        LONG par        // number of the concept’s parameter
);
```

Aim
Obtains the number of options of which the ‘par’ parameter consists of.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.
par: Number of the parameter. This should be a value between “0” and “n-1”, with “n” being the number of parameters within the concept.

Value returned
This returns the number of options for the “par” parameter. If an error is produced, the function will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcError().

```
LPCSTR EXPORT BdcParRotulo (
        HANDLE h,       // concept identifier
        LONG par        // number of the concept’s parameter
);
```

Aim
Obtains the label that identifies the “par” parameter for the concept.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.

par: Number of the parameter. This should be a value between “0” and “n-1”, with “n” being the number of parameters within the concept.

Value returned
Returns the label that identifies the “par” parameter for the concept, as a constant far pointer to a string. The function itself is responsible for allocating memory to the pointer. In case of error, the function will return “NULL”. To obtain more INFORMATION about the error produced, call the function BdcError().

```
LPCSTR EXPORT BdcOpcRotulo (
        HANDLE h,       // concept identifier
        LONG par,       // number of the concept's parameter
        LONG opt        // number of the parameter's option
);
```

Aim
Obtains the label that identifies the “opt” option of the concept’s “par” parameter.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.
par: Number of the parameter. This should be a value between “0” and “n-1”, with “n” being the number of parameters within the concept.
opt: Number of the parameter. This should be a value between “0” and “n-1”, with “n” being the number of options that the concept’s “par” parameter contains.

Value returned
Returns the label that identifies the “opt” option of the concept’s “par” parameter, as a constant far pointer to a string. The function itself is responsible for allocating memory to the pointer. In case of error, the function will return “NULL”. To obtain more INFORMATION about the error produced, call the function BdcError(). If referring to territorial scopes and currencies, see annexes 5 and 6.

3.2.2. Obtaining a comment.

```
LPCSTR EXPORT BdcComentario (
        HANDLE h,       // concept identifier
);
```

Aim
Obtains a comment text for the parametric concept.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.

Value returned
Returns the concept’s comment, as a constant far pointer to a string. The function itself is responsible for allocating memory to the pointer. In case of error, the function will return “NULL”. To obtain more INFORMATION about the error produced, call the function BdcError().

3.2.3. Assignation of parameter options and validation or calculation of the derivative.

BOOL EXPORT BdcValida (
HANDLE h, // concept identifier
LPLONG optl, // list of options for the parameters
);

Aim
To check whether a certain parametric combination is correct.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.
optl: Pointer to a vector (array) of LONGs with the options desired for each parameter. The options are numbered, starting from zero.

Value returned
Returns “0” if the combination is correct. If not, it will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcError().

BOOL EXPORT BdcCalcula (
HANDLE h, // concept identifier
LPLONG optl, // list of options for the parameters
);

Aim
To calculate the data corresponding to a parametric derivative.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.
optl: Pointer to a vector (array) of LONGs with the options desired for each parameter. The options are numbered, starting from zero.

Value returned
Returns “0” if executed correctly. In case of error, or if the combination is incorrect, returns “-1”. To obtain more INFORMATION about the error produced, call the function BdcError().

LONG EXPORT BdcValidos (
HANDLE h, // concept identifier
LPBYTE *optl, // list of parameter options for all valid derivative
);

Aim
Obtains the options for each parameter for all valid parametric derivative within the parametric family. If this function is not to be implemented in a specific family, it should return “0”. This is intended for families that possess a high number of possible combinations, for which determining which of these are valid through successive calls to the BdcCalcula() or BdcValida() functions would be very time consuming. In these cases, it is possible to obtain all valid combinations with a single call to this function.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.
optl: Pointer to a two-dimensional matrix (array) of BYTEs to be filled in for the function, in which each column corresponds to a valid parametric derivative, and each row corresponds to the values of each of the parameters. The matrix is returned in “C” style, in other words, in columns. The options are numbered, starting from zero. The function

itself is responsible for allocating memory to the pointer. The memory allocated in bytes will be the number of valid combinations multiplied by the number of parameters.

Value returned
Returns the number of valid parametric derivatives. If said information is not available, it will return “0”. In case of error, this will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcError().

Ordering
The valid combinations to be returned are ordered, bearing in mind that the first parameter has greater precedence than the second, with this being greater than the third, and so on.

Example
If a parametric family has 3 parameters, with 9, 10 and 11 options per parameter respectively, the total number of possible combinations is 9x10x11 = 990. If only the following four are valid (written in the order specified in the previous paragraph):
1. Values “0”, “5” and “9” of the parameters one to three, respectively.
2. Values “7”, “6” and “5” of the parameters one to three, respectively.
3. Values “8”, “2” and “4” of the parameters one to three, respectively.
4. Values “8”, “2” and “10” of the parameters one to three, respectively.

Then the function will return the following 4x3 = 12 bytes: 0, 5, 9, 7, 6, 5, 8, 2, 4, 8, 2 and 10.

3.2.4. Releasing memory.

```
LONG EXPORT BdcInValidos (
        HANDLE h,       // concept identifier
        LPBYTE *optl,   // list of parameter options for all invalid derivatives
);
```

Has the same aim and parameters as BdcValidos.
This function will be implemented instead of the latter when BdcValidos does not exist for a parametric concept and vice versa.

```
BOOL EXPORT BdcCierra (
        HANDLE h        // concept identifier
);
```

Aim
Closes the parametric concept and releases the allocated memory.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.

Value returned
Returns “0” if the operation is carried out correctly. In case of error, the function will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcError().

3.3. Accessible after BdcCalcula.

3.3.1. Obtaining the parametric derivative.

```
LONG EXPORT BdcDesNumero (
        HANDLE h        // concept identifier
);
```

Aim
Obtains the number of concepts into which the parametric derivative is decomposed.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.

Value returned
Returns the number of elements in its decomposition. A value of zero will indicate that the concept has no decomposition. It is possible that a single parametric concept may possess simple and compound derivatives. In case of error, the function will return "-1". To obtain more INFORMATION about the error produced, call the function BdcError().

```
LPCSTR EXPORT BdcDesCodigo (
        HANDLE h,       // concept identifier
        LONG des        // number of the decomposition's element
);
```

Aim
Obtains the CODE for the "des" number element into which the parametric derivative is decomposed.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.
des: Number of the decomposition's element for the concept. The elements are numbered, starting from zero.

Value returned
Returns the CODE for the number element "des" into which the parametric derivative is decomposed, as a constant far pointer to a string. The function itself is responsible for allocating memory to the pointer. In case of error, the function will return "NULL". To obtain more INFORMATION about the error produced, call the function BdcError().

```
BOOL EXPORT BdcFactor (
        HANDLE h,               // concept identifier
        LONG des,               // number of the decomposition's element
        double FAR *factor      // factor to be obtained
);
```

Aim
Obtains the output factor for the "des" number element into which the parametric derivative is decomposed.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.
des: Number of the decomposition's element for the concept. The elements are numbered, starting from zero.
*factor: Pointer in which the desired factor is returned – by default this is 1. The factor can be positive, zero or negative.

Value returned

Returns “0” if executed correctly. In case of error, the factor will be assigned to zero and the function will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcError().

```
BOOL EXPORT BdcRendimiento (
        HANDLE h,               // concept identifier
        LONG des,               // number of the decomposition’s element
        double FAR *ren         // output to be obtained
);
```

Aim
Obtains the output for the “des” number element into which the parametric derivative is decomposed.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.
des: Number of the decomposition’s element for the concept. The elements are numbered, starting from zero.
*ren: Pointer in which the desired output is returned. The output can be positive, zero or negative.

Value returned
Returns “0” if executed correctly. In case of error, the output will be assigned to zero and the function will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcError().

```
LPCSTR EXPORT BdcDesCodigoPorcentaje  (
        HANDLE h,       // concept identifier
        LONG des        // number of the decomposition’s element
);
```

Aim
Obtains the PERCENTAGE_CODEs applicable to the “des” number element in which the parametric derivative is decomposed.

Parameters
h: Identifier (HANDLE) of the parametric concept, which should be obtained in a prior call to the BdcLee() function.
des: Number of the decomposition’s element for the concept. The elements are numbered, starting from zero.

Value returned
Returns the PERCENTAGE_CODEs applicable for the number element “des” into which the parametric derivative is decomposed, as a constant far pointer to a string. The function itself is responsible for allocating memory to the pointer. The PERCENTAGE_CODEs will be separated by the character < ; > (ASCII-59) and the string may be returned empty if no percentage applies to the line. In case of error, the function will return “NULL”. To obtain more INFORMATION about the error produced, call the function BdcError().

```
BOOL EXPORT BdcPrecio (
        HANDLE h,               // concept identifier
        double FAR *pre         // unit price to be returned
);
```

Aim

Obtains the unit price if the parametric derivative is a simple derivative. It is possible that a single parametric concept may have both simple and compound derivatives.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.
*pre: Pointer in which the unit price is returned. Said price can be positive, zero or negative.

Value returned
Returns “0” if executed correctly. In case of error, the price will be assigned to zero and the function will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcError().

```
LPCSTR EXPORT BdcCodigo (
        HANDLE h        // concept identifier
);
```

Aim
Obtains the concept’s CODE.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.

Value returned
Returns the concept’s CODE, as a constant far pointer to a string. If a parametric derivative has been calculated (a call has been made to BdcCalcula), this CODE will be that of the parametric derivative. Otherwise, this will be the parametric concept’s CODE. The function itself is responsible for allocating memory to the pointer. In case of error, the function will return “NULL”. To obtain more INFORMATION about the error produced, call the function BdcError().

```
LPCSTR EXPORT BdcUnidad (
        HANDLE h        // concept identifier
);
```

Aim
Obtains the unit of measurement for the parametric derivative. This function allows a parametric concept to generate derived elements with different measurement units. For said function to work, the ~C registry must contain the special character “*” within the unit of measurement field. Said character indicates that the unit of measurement for the derived concepts is to be provided by the API. If the publisher of the DB wants the unit of measurement to be a value chosen by the user, they must also add the unit of measurement as another property of the parametric concept.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.

Value returned
Returns the text corresponding to the unit of measurement for the parametric derivative, as a constant far pointer to a string. The function itself is responsible for allocating memory to the pointer. If a summarized text has not been defined, the function will return the empty string “”. In case of error, the function will return “NULL”. To obtain more INFORMATION about the error produced, call the function BdcError().

LPCSTR EXPORT BdcResumen (
HANDLE h // concept identifier
);

Aim
Obtains the summarized text for the parametric derivative.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.

Value returned
Returns the summarized text for the parametric derivative, as a constant far pointer to a string. The function itself is responsible for allocating memory to the pointer. If a summarized text has not been defined, the function will return the empty string “”. In case of error, the function will return “NULL”. To obtain more INFORMATION about the error produced, call the function BdcError().

LPCSTR EXPORT BdcTexto (
HANDLE h // concept identifier
);

Aim
Obtains the complete text for the parametric derivative’s description.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.

Value returned
Returns the complete description text for the parametric derivative, as a constant far pointer to a string. The function itself is responsible for allocating memory to the pointer. If a complete text has not been defined, the function will return the empty string “”. In case of error, the function will return “NULL”. To obtain more INFORMATION about the error produced, call the function BdcError().

VOID EXPORT BdcTipoDescripcion (
LONG type // description type
);

Aim
Indicates the type of description to be obtained from a derived concept.

Parameters
Type: Refers to the different descriptions (obtained with BdcResumen and BdcTexto) that a derived concept can have according to the different contents of the database.
Value “0”: Default description for a parametric derivative.
Value “1”: Description of a parametric derivative into which one or various concept codes have been inserted and which are involved in its decomposition. Said codes appear indicated between dollar signs ($). This option allows programs to particularize the description of a derived concept when commercial products have been provided, associated to their components (with BdcComercNumero and BdcComercCodigo, or with the ~O registry).

Example: BdcTexto of the derived concept E612235K.
BdcTexto return with value 0 for BdcTipoDescripcion: Toilet with lid.
BdcTexto return with value 1 for BdcTipoDescripcion: Toilet $B1234567$ with lid $B7654321$.

```
LPCSTR EXPORT BdcPliego (
        HANDLE h,           // concept identifier
        LONG format,        // format identifier
        LONG type,          // specifies if the specification to be obtained is from the family
                            or the derivative
        LPCSTR section,     // code for the specification's section
        LPCSTR scope        // abbreviation of the scope
);
```

Aim
Obtains the specification text, according to model one.

Parameters

h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.

format: Text-format identifier. For now, this defines the following formats: "BDCFMT_ASCII" (if the specification is to be obtained in ASCII format), "BDCFMT_ANSI" (if the specification is to be obtained in ANSI format), "DCFMT_RTF" (if the specification is to be obtained in RTF format) and "BDCFMT_HTM" (if the specification is to be obtained in HTM format).

type: Specifies whether the common specification is to be obtained for the parametric family ("BDCPLI_FAMILIA" value) or for the derivative ("BDCPLI_DERIVADO" value). In the first case, it is not necessary to have previously carried out a call to BdcCalc BdcCalcula().

section: This can take the specification-section code whose text is to be obtained as the value, in which case it must be one of the sections specified in the ~L registry, as indicated in the "SPECIFICATION-TYPE REGISTRIES" section of the format's specifications. If no sections have been defined, the value of this parameter is not used. If, on the other hand, it takes the value NULL, the function will return the codes for the specification sections for which the concept has specification text, separated by the subfield separator ("\").

scope: Scope from which the specification text is to be obtained. Corresponds to one of the "SCOPE_ABREV" fields specified in the registry ~W, as indicated in the "GEOGRAPHIC-SCOPE-TYPE REGISTRY" section of the format's specifications. If no scopes have been defined (there is no ~W registry), the value of this parameter is ignored. In this case, it is possible that the scope may be a global parameter for the database.

Value returned
If the section parameter contains the code for the specification section whose text is to be obtained, the function will return the parametric derivative's specification text in the requested format, as a constant "far" pointer to a string. If a specification text has not been defined, the function will return the empty string "".
When the parameter section is NULL, the function will return the labels of the existing specifications associated with the parametric concept in the format {label\}, if there are no labels associated with the concept the function will return "".
The function itself is responsible for allocating memory to the pointer. In case of error (for example a format not supported by the BDC), the function will return NULL. To obtain more INFORMATION about the error produced, call the function BdcError().

```
LPCSTR EXPORT BdcCodParrafo (
        HANDLE h,           // concept identifier
        LONG type,          // specifies if the specification to be obtained is that of the
                            family or the derivative
        LPCSTR section,     // code for the specification's section
```

```
        LPCSTR scope            // abbreviation of the scope
);
```

Aim
Obtains the codes for the specification paragraphs, according to model two.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.
type: Specifies whether the common specification is to be obtained for the parametric family ("BDCPLI_FAMILIA" value) or for the derivative ("BDCPLI_DERIVADO" value). In the first case, it is not necessary to have previously carried out a call to BdcCalcula().
section: Code for the specification section whose paragraph codes are to be obtained. This must be one of the sections specified in the ~L registry, as indicated in the "SPECIFICATION-TYPE REGISTRIES" section of the format's specifications. If no sections have been defined, the value of this parameter is not used.
scope: Scope from which the paragraph codes are to be obtained. Corresponds to one of the "SCOPE_ABREV" fields specified in the registry ~W, as indicated in the "GEOGRAPHIC-SCOPE-TYPE REGISTRY" section of the format's specifications. If no scopes have been defined (there is no ~W registry), the value of this parameter is ignored. In this case, it is possible that the scope may be a global parameter for the database.

Value returned
Returns the text with the specification-paragraph codes for the parametric derivative, as a constant "far" pointer to a string. The function itself is responsible for allocating memory to the pointer. If no paragraph code has been defined, the function will return the empty string "". In case of error (for example a scope not supported by the BDC), the function will return NULL. To obtain more INFORMATION about the error produced, call the function BdcError(). The codes will be separated with the usual subfield separator, in other words, the text returned will have the syntax { PARAGRAPH_CODE \ }

```
LPCSTR EXPORT BdcTexParrafo (
        LONG format             // format identifier
        LPCSTR paragraph_cod    // specification-paragraph code
);
```

Aim
Obtains the specification text for the parametric derivative corresponding to the paragraph code 'paragraph_cod', according to model two. The paragraph code is obtained through a call to the function BdcCodParrafo.

Parameters
format: Text-format identifier. For now, this defines the following formats: "BDCFMT_ASCII" (if the specification is to be obtained in ASCII format), "BDCFMT_RTF" (if the specification is to be obtained in RTF format) and "BDCFMT_HTM" (if the specification is to be obtained in HTM format).
paragraph_cod: Paragraph code of the specification whose text is intended to be obtained.

Value returned
Returns the paragraph text in the requested format, as a constant "far" pointer to a string. The function itself is responsible for allocating memory to the pointer. This memory is released in the following call to this function within the same process. If a specification text has not been defined with this paragraph code, the function will return the empty string "". In case of error (for example a format not supported by the BDC), the function will return NULL. To obtain more INFORMATION about the error produced, call the function BdcError().

```
LPCSTR EXPORT BdcClaves (
        HANDLE h        // concept identifier
);
```

Aim
Obtains the parametric derivative's thesaurus keys.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.

Value returned
Returns the parametric derivative's thesaurus keys, as a constant "far" pointer to a string, with the same format as the ~A registry, in other words, "{THESAURUS_KEY\}". The function itself is responsible for allocating memory to the pointer. If no thesaurus keys have been defined, the function will return the empty string "". In case of error, the function will return "NULL". To obtain more INFORMATION about the error produced, call the function BdcError().

```
LPCSTR EXPORT BdcTipo (
        HANDLE h,               // concept identifier
);
```

Aim
Obtains the concept type.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.

Value returned
Returns the concept's type, as a constant "far" pointer to a string. If a parametric derivative has been calculated (a call has been made to the function BdcCalcula), this type will be that of the parametric derivative. Otherwise, this will be the type of the parametric concept. The function itself is responsible for allocating memory to the pointer. In case of error, the function will return "NULL". To obtain more INFORMATION about the error produced, call the function BdcError(). Corresponds to the TYPE field of the ~C registry.

```
LONG EXPORT BdcComercNumero(
        HANDLE  h                       // concept identifier
        LPCSTR   DB_root_code           // identifier of the entity creating the DB
);
```

Aim
Obtains the number of concepts linked to a concept belonging to DB_root_code and used by the latter in its coding classification system.

Parameters
h: Identifier (HANDLE) for the parametric concept.
DB_root_code: Refers to the identification of the CODE for the entity creating the DB. This CODE must be provided by the entity that prepares the DB, to avoid ambiguities. It is recommended that this be the entity's own VAT number.

Value returned
Returns the number of linked concepts. If a parametric derivative has been calculated (a call has been made to the function BdcCalcula), this number will be that of the parametric derivative.

Otherwise, this will be the number of the parametric concept. A value of zero means that there are no linked concepts. In case of error, the function will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcError().

```
BOOL EXPORT BdcComercCodigo(
        HANDLE  h                       // concept identifier
        LONG commerc                    // number of the linked concept
        LPLCSTR *File_code              // name of the file to be returned
        LPLCSTR *Entity_Code /          / identifier of the code of the entity to which
                                        // the information to be returned is associated
        LPLCSTR *Concept_Code           // concept that belongs to the Entity_Code to be
returned
);
```

Aim
Obtains the file code that indicates where the information referring to entity_code#concept_code can be found, or the entity_code and the concept_code that can be found in the same DB.

Parameters
h: Identifier (HANDLE) for the parametric concept.
commerc: Number of the linked concept. The linked concepts are numbered starting from zero.
file_code: Refers to the name of the file that, if it exists, indicates the place where the information referring to entity_code#concept_code is located. If the value is null, the entity code and concept code fields should exist. The function itself is responsible for allocating memory to the pointer.
entity_code: Refers to the identification of the CODE for the entity to which INFORMATION is associated. This CODE must be provided by the entity that prepares the DB, in accordance with its classification system, to avoid ambiguities. It is recommended that this be the entity's own VAT number. The function itself is responsible for allocating memory to the pointer.
concept_code: Refers to a concept belonging to ENTITY_CODE and used by the DB-compiling entity. When CONCEPT_CODE refers to a commercial product, such CODE shall be provided by the manufacturer and can never coincide with the designation of DB_ROOT_CODE, ENTITY_CODE or CONCEPT_CODE when it refers to a generic concept. The function itself is responsible for allocating memory to the pointer.
As this commercial product has been treated as a CONCEPT, it can use all registries existing in the format to specify its associated INFORMATION (price, graphic INFORMATION, etc.). To be able to use the registries, the concept’s identifier code shall be ENTITY_CODE # CONCEPT_CODE.

Value returned
Returns “0” if executed correctly. In case of error, the function will return “-1”. To obtain more INFORMATION about the error produced, call the function BdcError().

```
LONG EXPORT BdcDocNumero (
        HANDLE  h       // concept identifier
);
```

Aim
Obtains the number of files associated with a concept.

Parameters
h: Concept’s identifier (HANDLE).

Value returned
Returns the number of associated files. A value of zero means that there are no associated files. In case of error, the function will return “-1”. To obtain more INFORMATION about the error

produced, call the function BdcError().

```
BOOL EXPORT BdcDocCodigo (
        HANDLE  h,              // concept identifier.
        LONG num,               // number of the associated file.
        LONG * type             // classification of the file's information type.
        LPCSTR * file,          // name of the file containing concept information.
        LPCSTR * description    // description of the information contained in the file.
);
```

Aim
Obtains the file associated to a concept, as well as its thematic classification and associated description. The behaviour will be the same as the discrete: first it will search for the file in the local directory, and if it is not there, in the URL_BASE+file. URL_BASE is defined in registry ~V.

Parameters
h: Concept's identifier (HANDLE).
num: Number of the file associated with the concept. These are numbered starting from zero.
type: Returns the code of one of the types numbered in registry ~F.
file: Returns the name of the file with the extension containing the concept information. The file may contain a URL_EXT. The extensions permitted are those specified in registry ~F. If various files are returned, the first of the files is the principal one and the rest are linked. In this case, the format of the string will be as follows: "file 1 | ... | file n | "
description: Returns a brief description of the information contained in the file. Returns NULL if there are no associated descriptive texts.

Value returned
Returns "0" if executed correctly. In case of error, the function will return "-1". To obtain more INFORMATION about the error produced, call the function BdcError().

```
LONG EXPORT BdcNumProp (
HANDLE h,       // concept identifier
LONG use        // application for which the number of technical properties is required

);
```

Aim
Obtains the number of technical properties defined for a concept for the use requested. Initially, use 0 is defined for the calculation of energy costs, CO2 emissions or waste.

Parameters
h: Parametric-concept identifier (HANDLE), which should be obtained in a prior call to the BdcLee() function.
Use: Number that indicates which calculation is being defined.

Value returned
Number of technical properties defined for the parametric concept for this use.

```
BOOL EXPORT BdcPropValString (
HANDLE h,               // concept identifier
LONG use,               // application for which the number of technical properties is required
LONG nprop,             // property number
LPCSTR *property,       // pointer to property number
LPCSTR *value,          // pointer to property value
```

```
LPCSTR *um              // pointer to the property's unit of measurement
);
```

Aim
Obtains the number of the technical property, its value, and the unit of measurement.

Parameters
h: Identifier (HANDLE) for the parametric concept.
use: Number that indicates which calculation is being defined.
nprop: Property number. These are numbered starting from zero.
property: Pointer to the IT_CODE of one of the properties numbered in the registry ~X.
value: Pointer to the value of the property, which can be alphabetic or numeric.
um: Pointer to the property's unit of measurement If the values of the property are numerical values, it shall be indicated according to the International System of Units of Measurement (see Annex 7).

Value returned
Returns "0" if executed correctly. To obtain more INFORMATION about the error produced, call the function BdcError().

```
LONG EXPORT BdcNumComponentes (
        HANDLE h,           // concept identifier
        LONG type           // decomposition type
);
```

Aim
Obtains the number of components for a concept, for each type of decomposition.

Parameters
h: Identifier (HANDLE) for the parametric concept.
type: Number of the type of decomposition described in the field DECOMPOSITION_TYPE in the registry ~R.

Value returned
Returns the number of components for the relation type.
If the value returned = 0, the concept has no components of this type.
If the value returned = -1, the concept has incomplete information.

```
BOOL EXPORT BdcCodigoComponente (
        HANDLE h,           // concept identifier
        LONG type,          // decomposition type
        LONG ncomp,         // component number for the standard decomposition for
                            // the concept
        LPCSTR *code        // pointer to the component code
);
```

Aim
Obtains the component code for an element ("ncomp" number) for each type of decomposition.

Parameters
h: Identifier (HANDLE) for the parametric concept.
type: Number of the type of decomposition described in the field DECOMPOSITION_TYPE in the registry ~R.
ncomp: Component number for the standard concept relation. These are numbered starting from zero.
Code: Pointer to the component code.

Value returned
Returns “0” if executed correctly. To obtain more INFORMATION about the error produced, call the function BdcError().

```
LONG Export BdcNumPropComponente (
        HANDLE h,           // concept identifier
        LONG type,          // decomposition type
        LONG ncomp          // component number
);
```

Aim
Obtains the number of properties for a component, for each type of decomposition.

Parameters
h: Identifier (HANDLE) for the parametric concept.
type: Number of the type of decomposition described in the field DECOMPOSITION_TYPE in the registry ~R.
ncomp: Component number for the standard concept relation. These are numbered starting from zero.

Value returned
Number of properties for the relation.

```
BOOL EXPORT BdcComponentePropValString (
        HANDLE h,           // concept identifier
        LONG type,          // decomposition type
        LONG ncomp,         // component number
        LONG nprop,         // property number
        LPCSTR *property,   // pointer to property number
        LPCSTR *value,      // pointer to property value
        LPCSTR *um          // pointer to the property’s unit of measurement
);
```

Aim
Obtains the number of the “nprop” property, its value, and the unit of measurement.

Parameters
h: Identifier (HANDLE) for the parametric concept.
type: Number of the type of decomposition described in the field DECOMPOSITION_TYPE in the registry ~R.
ncomp: Component number for the standard concept relation. These are numbered starting from zero.
nprop: Property number. These are numbered starting from zero.
property: Pointer to the name of the property’s relation.
value: Pointer to the value of the property, which can be alphabetic or numeric.
um: Pointer to the property’s unit of measurement If the values of the property are numerical values, it shall be indicated according to the International System of Units of Measurement (see Annex 7).

Value returned
Returns “0” if executed correctly. To obtain more INFORMATION about the error produced, call the function BdcError().

4. ERROR MESSAGES.

The error CODES are stored in a LONG (32-bit integer) so that each error corresponds to one bit. This way, it is possible to define up to 32 error CODES that may be produced alone or together. Calls to the functions BdcGloError() and BdcError() remove the error CODES previously produced.

For example, to find out if a specific error has been produced with the “Concept” concept, the following syntax should be used:

```
const char *Message.
LONG cod_err = BdcError ((HANDLE)Concept, &Message);
if (cod_err & BDCERR_ BASE_DATOS) {
        // The “BDCERR_ BASE_DATOS” error has been produced
        ...
}
```

4.1. Error message CODES.

BDCERR_CORRECTO	There is no error.
BDCERR_BASE_DATOS	An error message exists. In this case, the parametric DEFINITION indicates an invalid combination and returns an explanatory error message.
BDCERR_PARAMETRO	A non-existent parameter is passed to BdcCalcula or BdcGloCalcula.
BDCERR_OPCION	A non-existent option is passed to BdcCalcula or BdcGloCalcula.
BDCERR_MAX_OPCIONES	More than 62 options are defined in a specific parameter.
BDCERR_NO_LEIDO	The user has tried to use BdcCalcula() before BdcLee().
BDCERR_NO_CALCULADO	The user has tried to access the data of a parametric derivative before using BdcCalcula().
BDCERR_DESCOMPOSICION	The user has tried to access an element for a non-existent decomposition.
BDCERR_SIN_CODIGO	No CODE has been defined.
BDCERR_SIN_MEMORIA	Insufficient memory.
BDCERR_CONCEPTO_NULO	A void HANDLE was passed.
BDCERR_FMT_NO_SOPORTADO	The format of the text requested is not supported by the BDC.
BDCERR_NO_COMBINACION	A non-existent combination was passed to the BdcValida or BdcCalcula call.
BDCERR_NO_BANCO_PRECIOS	No Price Bank exists.
BDCERR_PARAMETRO_INCORRECTO	“The program has requested an incorrect parameter”

EXAMPLES.

1. DATABASE EXAMPLE: BASE.DLL

In the files that accompany this document (and which can be found on the association's website, "http://www.fiebdc.es"), both the files that make up the database and the necessary sources for its construction (compilation) are made available.

The example developed has the following characteristics (not all databases made according to this format will have the same characteristics):

1. An independent parameter-codification model has been used.
2. Specification-text model one has been used.
3. A global parametric exists, although the value of its sole parameter is only used in the concept "SBRG.1$".
4. A concept ("Esp$") exists which does not respond to the FIEBDC-3/95 criteria: it possesses more than four parameters, and the family code only has four characters.
5. All parametric concepts have a CODE ending in "$", although the format does not require it.
6. A concept ("SBRG.1$") exists whose derivatives possess a code that does not respond to the FIEBDC-3/95 criteria. To allocate a code, the technique for the definition of a table of synonyms has been followed: as such, the concept obtained through choosing the first value from the only parameter it possesses will have the code "SBRG.1_18" instead of the code "SBRG.1a".
7. The specification texts are only defined in ASCII format and for the parametric derivatives. They are not divided into sections (facets) or scopes.

1.1. Files for the Database's distribution.

To distribute the database constructed with this example, the following files must be provided:

base.bc3: ASCII database file in FIEBDC-3/98 format. In the example, the database includes its global parametric in this DLL, as well as parametric descriptions for the concepts "ABPH.1$", "SBRG.1$", "EADR.3$" and "Esp$", and as such must include at least the following registries:

```
~P| | | BASE.DLL |
~P| ABPH.1$ | |
~P| SBRG.1$ | |
~P| EADR.3$ | |
~P| Esp$ | |
```

base.dll: DLL which contains the parametric descriptions and the STANDARD interface with applications (API).

1.2. Files required for the Database's CONSTRUCTION.

This example is prepared to be compiled with the Microsoft Visual Studio 2010 multibyte set of characters as a 32 and 64-bit DLL.

To construct the DLL, the following files are required:

fiebdc.h: File defining the format.
base.h: DEFINITION of variables and useful outlines for the definition of the parametric descriptions.
interfaz.cpp: Implementation of the API's functions.
aplicat.cpp: Implementation of the functions of the parametric description.
base.cpp: Implementation of the parametric descriptions for the database in C++ format. This is the only section written by the database's publishers. A syntax like the

parametric description of the FIEBDC-3/95 format has been used, to facilitate exchanges between both formats.

base.def: DEFINITION of the API's export functions.

2. EXAMPLE APPLICATION: PROGRAMA.EXE

In the files that accompany this document (and which can be found on the association's website, "http://www.fiebdc.es"), both the executable ("Programa.EXE") and the necessary sources for its construction (compilation) are made available.

This simple application reads the concepts included in the source of the BASE.DLL database itself (in a real application, the concepts are defined in the ~P registries of base.bc3) and writes the labels of all parameters as well as all data regarding their parametric combinations in the file "SALIDA.TXT".

The sources are easily modifiable to be able to use the program to test any database that uses parametrics compiled in DLL, according to the API established in this document.

2.1. Files required for the CONSTRUCTION of the example program.

This example is prepared to be compiled with the Microsoft Visual Studio 2010 multibyte set of characters as a 32 and 64-bit Windows applicable in console mode.

To construct the application, the following files are required:

fiebdc.h:	File defining the format.
program.h:	Declaration of variables and functions.
program0.c:	Auxiliary functions for the database's opening and closing and the processing of error messages.
program.c:	Example program.

Annex 4. Classification in types of Concepts.

1) Types 0, 1, 2, 3, 4 and 5.

These types correspond to those in force in the FIEBDC-3/2016 format.

0	Unclassified
1	Labour
2	Machinery
3	Materials
4	Additional-waste components
5	Waste classification

2) Types obtained in the Boletín Oficial del Estado price-revision indexes and polynomial formulas according to the official indexes of RD 1359/2011.

This classification is derived from the Boletín Oficial del Estado (www.boe.es), according to the official indexes of Real Decreto 1359/2011, applicable for public-works contracts awarded from 1 August 2013 for works for the public administration, to establish the price-variation indexes through their respective polynomial formulas.

A	Aluminium
B	Bituminous materials
C	Cement
E	Energy
F	Spotlights and luminaires
L	Ceramic materials
M	Wood
O	Floors
P	Plastic products
Q	Chemical products
R	Aggregates and rock
S	Steel materials
T	Electronic materials
U	Copper
V	Glass
X	Explosive materials

To the above, we add the following:

1	Labour
2	Machinery
3	Other materials
%	Auxiliary equipment

Where,

1	Refers to type 1 of the classification “Types 0, 1, 2, 3, 4 and 5”.
2	Refers to type 2 of the classification “Types 0, 1, 2, 3, 4 and 5”.
A, B, C, E, F, L, M, O, P, Q, R, S, T, U, V, X and 3	refer to type 3 of the classification “Types 0, 1, 2, 3, 4 and 5”. Said classification of materials corresponds to that of the price-revision indexes for official works, to which we add type M for those materials that cannot be associated with any of the above.
%	This refers to a new type which corresponds to the auxiliary equipment that may appear in a price justification.

3) Types obtained in the Boletín Oficial del Estado price-revision indexes and polynomial formulas according to the official indexes prior to RD 1359/2011 and the CNC.

This classification is derived from the Boletín Oficial del Estado (www.boe.es), according to the official indexes of Real Decreto 1359/2011, applicable for public-works contracts awarded until 31 July 2013 for works for the public administration, and of the CNC (Confederación Nacional de la Construcción: www.cnc.es), in order to establish the price-variation indexes through their respective polynomial formulas.

H	Labour
MC	Cement
MCr	Ceramics
MM	Woods
MS	Steels
ME	Energy
Mcu	Copper
Mal	Aluminium
ML	Binders

To the above, we add the following:

M	Other materials
Q	Machinery
%	Auxiliary equipment

Where,

H Refers to type 1 of the classification “Types 0, 1, 2, 3, 4 and 5”.

MC, MCr, MM, MS, ME, MCu, MAl, ML and M refer to type 3 of the classification “Types 0, 1, 2, 3, 4 and 5”. Said classification of materials corresponds to that of the price-revision indexes for official works, to which we add type M for those materials that cannot be associated with any of the above.

Q Refers to type 2 of the classification “Types 0, 1, 2, 3, 4 and 5”.

% This refers to a new type which corresponds to the auxiliary equipment that may appear in a price justification.

4) Types obtained from the Asociación de Redactores de Bases de Datos de Construcción.

These types serve to develop type 0 of the classification “Types 0, 1, 2, 3, 4 and 5”.

EA	Auxiliary element
EU	Unitary element
EC	Compound element
EF	Functional element
OB	Work
PA	Lump Item
PU	Unitary Budget

Where,

Auxiliary element: Constructive element formed of a combination of basic elements (labour, materials, and machinery) involved in the formulation of a unit of work.
Examples: H-250 concrete of plastic consistency produced on site; AEH-400 S steel produced on site; etc.

Unitary element: Constructive element formed of a set of basic and/or auxiliary elements that make up a unit of work and which are carried out by the same group of specialists.
Examples: Partition wall of 25x12x4 cm simple ceramic bricks, with 1:6 cement mortar; H-250 concrete in supports, etc.

Compound element: Constructive element formed of a set of basic, auxiliary and/or unitary elements that constitute a constructive complex and which is carried out by one or several groups of specialists.
Examples: H-250 cement reinforced with 120 kg of AEH-400 S steel and formwork with metal plates, on supports […].; Facade enclosure formed of two ceramic sheets with air-chamber insulation [..]; etc.

Functional element: Constructive element formed of a set of basic, auxiliary, unitary and/or compound elements that constitute a constructive complex with a complete function within the work.
Examples: Reinforced cement structure; Kitchen formed of [...]; etc.

Work: Constructive element formed of a complex of functional, compound, unitary, auxiliary and/or individual elements that make up the totality of elements constituting a construction.
Examples: Facade restoration [...]; Construction of multi-family housing [...]; etc.

Lump Item: Unit of work to be justified.
Example: Lump item to be accounted for representing 1% of the BME, for cultural-action expenses.

Unitary Budget: Concept that refers to a partial budget composed of units of work. It is like a Functional Element (FE) but behaves differently in its application. Its behaviour is such that it may appear as a unit of work in a budget to which a measurement is assigned; while, on the other hand, it behaves as a budget subchapter in that no indirect costs are to be applied to it (indirect costs, if any, are already covered by the units of work in its composition) and do not appear in the price tables 1 and 2.

Annex 5. Territorial Scopes.

As abbreviations of Territorial Scopes corresponding to provinces and autonomous communities of the Spanish state, for their possible use in the IDENTIFICATION_LABEL field of the ~V registry and/or the SCOPE_ABREV field of the registry ~W, the following are established:

E	Spain		
	AND		Autonomous Community of Andalusia
		AL	Almeria
		CO	Cordoba
		H	Huelva
		CA	Cadiz
		GR	Granada
		J	Jaen
		MA	Malaga
		SE	Seville
	ARA		Autonomous Community of Aragon
		TE	Teruel
		HU	Huesca
		Z	Zaragoza
	AST		Autonomous Community of the Principality of Asturias
		O	Asturias
	BAL		Autonomous Community of the Balearic Islands
		PM	Balearic Islands
	CAN		Autonomous Community of the Canary Islands
		GC	Las Palmas
		TF	Tenerife
	CBR		Autonomous Community of Cantabria
		S	Cantabria
	CLM		Autonomous Community of Castile-La Mancha
		AB	Albacete
		CR	Ciudad Real
		CU	Cuenca
		GU	Guadalajara
		TO	Toledo
	CAL		Autonomous Community of Castile and Leon
		AV	Avila
		SG	Segovia
		SO	Soria
		VA	Valladolid
		BU	Burgos
		LE	Leon
		P	Palencia
		SA	Salamanca
		ZA	Zamora
	CAT		Autonomous Community of Catalonia
		B	Barcelona
		GI	Girona
		T	Tarragona
		L	Lleida
	EXT		Autonomous Community of Extremadura
		BA	Badajoz
		CC	Caceres
	GAL		Autonomous Community of Galicia
		LU	Lugo
		OR	Ourense
		PO	Pontevedra
		C	A Coruña
	MAD		Community of Madrid

	M	Madrid
MUR		Autonomous Community of the Region of Murcia
	MU	Murcia
NAV		Chartered Community of Navarre
	NA	Navarre
PVA		Autonomous Community of the Basque Country
	VI	Alava
	BI	Bizkaia
	SS	Guipuzkoa
RIO		Autonomous Community of La Rioja
	LO	La Rioja
VAL		Valencian Community
	V	Valencia
	A	Alicante
	CS	Castellon

Annex 6. Currencies.

As abbreviations of Currencies (CURRENCY field of the ~K registry), those specified by ISO 4217 are established. A few are attached by way of example:

Monetary Unit	**European Monetary Union**
ATS	Austrian Schilling
BEF	Belgian Franc
DEM	Deutsche Mark
ESP	Spanish Peseta
FIM	Finnish Markka
FRF	French Franc
GRD	Greek Drachma
IEP	Irish Punt
ITL	Italian Lira
LUF	Luxembourg Franc
NLG	Dutch Guilder
PTE	Portuguese Escudo

Non EMU currency	**Others**
AUD	Australian Dollar
BGN	Bulgarian Lev
CAD	Canadian Dollar
CHF	Swiss Franc
CYP	Cypriot Pound
CZK	Czech Krona
DKK	Danish Krone
EEK	Estonian Kroon
EUR	Euro
GBP	Pound Sterling
HKD	Hong-Kong Dollar
HUF	Hungarian Forint
ISK	Icelandic Krona
JPY	Japanese Yen
KRW	South-Korean Won
LTL	Lithuanian Litas
LVL	Latvian Lats
MTL	Maltese Lira
NOK	Norwegian Krone
NZD	New Zealand Dollar
PLN	Polish Zloty
ROL	Romanian Leu
SEK	Swedish Krona
SGD	Singapore Dollar
SIT	Slovenian Tolar
SKK	Slovakian Koruna
TRL	Turkish Lira
USD	US Dollar
ZAR	South-African Rand

Annex 7. Units of Measurement.

“The obligatory Legal System of Units of Measurement in Spain is the decimal metric system of seven basic units, known as the International System of Units (IS) adopted by the General Conference on Weights and Measurements and applicable in the European Economic Community”. RD 2032/2009, of 30 December, through which the legal units of measurements are established.

The nomenclature of the following units has been adopted in accordance with this Real Decreto:

m	Metre
m2	Metre squared
m3	Metre cubed
kg	Kilogram
km	Kilometre
t	Tonne
l	Litre
h	Hour
d	Day
to	Area
ha	Hectare
cm3	Centimetre cubed
cm2	Centimetre squared
dm3	Decimetre cubed

Due to similarities with said Real Decreto, and derived from the Asociación de Redactores de Bases de Datos de Construcción, the following have also been adopted:

u	Unit
mu	Thousand units
cu	Hundred units
month	Month

Annex 8. Definitions of different types of Budgets.

The BUDGET is the sum of the quantities obtained in the measurement of units of work through their respective prices, organized according to a structure (example: root, chapters, subchapters, and items).

BUDGET OF MATERIAL EXECUTION (BME)

Article 131 of Reglamento General de la Ley de Contratos de las Administraciones Públicas:

“The material execution budget is the result obtained from the sum of the products of the number of each unit of work, multiplied by its unit price and the lump items.”

Article 130. Calculation of the prices of different units of work:

“1. The calculation of the prices of different units of work will be based on the determination of precise direct and indirect costs for its execution, without incorporating – in any case – the amount of the VAT that may be taxed on the delivery of goods or services carried out.

2. The following are considered direct costs:
a) The labour directly involved in the unit of work’s execution.
b) The materials, at the resulting on-site prices, integrated within the unit in question or which may be necessary for its execution.
c) The staffing, fuel, and energy expenses, among others, incurred in the operation of machinery and facilities used in the execution of the unit of work.
d) The depreciation and storage costs of the machinery and facilities.

3. The following are considered indirect costs:
The costs for the installation of on-site offices; communications; the building of warehouses, workshops, or temporary halls for workers; laboratories, etc.; staffing costs for technical and administrative staff assigned exclusively to the work; and contingencies. All these expenses, except those reflected in the budget valued in units of work or in lump items, will be represented in a percentage of the direct costs, as well as for all units of work, to be adopted in all cases by the author of the project considering the nature of the work planned, the size of the budget and the foreseeable completion time.

4. In those cases in which unforeseen price fluctuations, after the projects’ approval, render the price calculations appearing in their budgets outdated, the contracting bodies can proceed to update these if the work is of an urgent nature, by applying a lineal percentage of the increase, to adjust the expressed prices to those in force on the market at the time of the tender.

5. The contracting authorities dictate the additional instructions applicable to the calculation of unit prices in different projects carried out by their departments.”

Article 154. Lump items:

“1. The lump items will be valued in accordance with that indicated in the specific technical specifications. Failing this, the following will be considered:
a) As lump items to be accounted for, those which can be measured in all their parts within units of work, with unit prices, and
b) As full-payment lump items, those referring to work whose specification appears in the project’s contractual documents and which cannot be measured according to the statement.

2. The lump items to be justified will be valued at the award prices according to the conditions of the contract and the result of the corresponding measurements. When the prices of one or several units of work are not included in the price table, a procedure will be carried out in accordance with that provided in article 146.2 of the Law, in which case – for the introduction of new prices determined as such – the following two conditions must be met:

a) The contracting body has approved the justification and decomposition of the budget for the lump item, in addition to the new prices, and

b) The total amount of said lump item, considering both the prices included in the price table and the new applicable prices in its valuation, does not exceed the amount of the lump item appearing in the project.

3. The full-payment lump items will be paid in full to the contractor, once the work or works to which they refer have been determined, according to the contract conditions and without prejudice to what the specific administrative clauses may establish with respect to their payment in instalments in justified cases.
When the specification of the work or works constituting a full-payment lump item do not appear in the project's contractual documents or are incomplete, imprecise, or insufficient for the purposes of their execution, the instructions issued in writing to this effect by the management shall be complied with, which the contractor can oppose in case of disagreement."

TENDER-BASE BUDGET
(formerly known as Contractual-Execution Budget CEB).

Article 131 of Reglamento General de la Ley de Contratos de las Administraciones Públicas:

"The tender-base budget will be obtained by increasing that of the material execution in the following concepts:

1. General structural expenses affecting the contract, calculated in the following percentages applied to the material execution budget:

a) From 13 to 17%, to be established by each Ministerial Department, in view of concurrent circumstances, for the company's general expenses, financial expenses, fiscal charges (VAT exclusive) and legally established administration fees that affect the cost of the works and other expenses derived from contract obligations. Likewise, taxes that may be applied to the tax on natural and legal persons will be excluded.

b) 6% of the contractor's industrial profit.

These percentages can be modified, in general, by agreement of the Comisión Delegada del Gobierno para Asuntos Económicos when, because of variation in current budgets, it is deemed necessary.

2. The VAT taxed on the works' execution, whose rate will be applied to the sum of the material execution budget and the general structural expenses outlined in section 1."

VAT may vary depending on geographic area, in which case the corresponding tax will be imposed – for example, the Canary Islands General Indirect Tax (Spanish IGIC).

In private works, the percentages of the company's general expenses and the contractor's industrial profit may differ from those previously stated.

TENDER BUDGET (TB).

Amount that serves as the base upon which to formulate the financial offers provided by the companies competing in a tender.

AWARD BUDGET (AB).

This is the amount corresponding to the financial proposal of the company awarded the contract. This may be the same as the Tender Budget, less than this if there is a decrease, or more than this if any increases arise.

It is recommended that the increase or decrease is applied to the Material Execution budget, as the resulting quantity will be the same, without reflecting modifications in the allocation of VAT.

Annex 9. Criteria for the assignment of references in IFC to price banks in FIEBDC format.

In recent years, the assignment of codes to the elements of BIM models is becoming increasingly common, to refer to concepts existing within price bases in BC3 format.

To standardize the use of these codes, the Standing Technical Committee of the FIEBDC Association proposes the use of the following criteria to users of BIM programs:

- Include in each BIM element a parameter or attribute for each text type, with the name “BC3”. The concept code for the price base (up to 20 characters) will be entered in this parameter. If several database concepts need to be allocated to a single BIM element, all these can be entered in this parameter, separated by commas. In the case of several codes, the order of the codes for vertical elements shall go from inside to outside, and for horizontal elements from above to below.

- Optionally, the parameter can also be included, named “BC3_URL” with the access URL address for the information regarding this price-base concept, in BC3 format. If the case of several concepts, various comma-separated URL addresses can be included. If any of these are not available, it will be left blank and only separated by commas.

- Optionally, it can also include the parameter “BC3_BASE” with the name of the price base used, as well as its year or version. In the case of several concepts, these can be included in various comma-separated databases. If any of these are not available, it will be left blank and only separated by commas.

On the other hand, if the BIM model is to be linked with a budget in BC3 format, another global parameter can be included, named “BC3_BUDGET_URL”, including the URL in which the budget file can be found in BC3 format.