BS EN 62258-2:2011

BSI Standards Publication

Semiconductor die products

Part 2: Exchange data formats

...making excellence a habit.™

National foreword

This British Standard is the UK implementation of EN 62258-2:2011. It is identical to IEC 62258-2:2011. It supersedes BS EN 62258-2:2005 which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee EPL/47, Semiconductors.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© BSI 2011

ISBN 978 0 580 61892 5

ICS 31.080.99

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 July 2011.

Amendments issued since publication

Amd. No.	Date	Text affected

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 62258-2

July 2011

ICS 31.080.99

Supersedes EN 62258-2:2005

English version

Semiconductor die products - Part 2: Exchange data formats (IEC 62258-2:2011)

Produits de puces de semiconducteurs -
Partie 2: Formats d'échange de données
(CEI 62258-2:2011)

Halbleiter-Chip-Erzeugnisse -
Teil 2: Datenaustausch-Formate
(IEC 62258-2:2011)

CENELEC

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

Management Centre: Avenue Marnix 17, B - 1000 Brussels

Foreword

The text of document 47/2085/FDIS, future edition 2 of IEC 62258-2, prepared by IEC TC 47, Semiconductor devices, was submitted to the IEC-CENELEC parallel vote and was approved by CENELEC as EN 62258-2 on 2011-06-29.

This European Standard supersedes EN 62258-2:2005.

With respect to EN 62258-2:2005, the following parameters have been updated for EN 62258-2:2011:

Subclause	**Parameter name**
8.2.9	DEVICE_PICTURE_FILE
8.2.10	DEVICE_DATA_FILE
8.4.6	TERMINAL_GROUP
8.4.7	PERMUTABLE
8.5.1	TERMINAL_MATERIAL (was DIE_TERMINAL_MATERIAL)
8.5.2	TERMINAL_MATERIAL_STRUCTURE
8.6.2	MAX_TEMP_TIME
8.7.6	SIMULATOR_*simulator*_TERM_GROUP
8.8.3	ASSEMBLY
8.9.2	WAFER_THICKNESS
8.9.3	WAFER_THICKNESS_TOLERANCE
8.9.9	WAFER_INK
8.10.4	BUMP_SHAPE
8.10.5	BUMP_SIZE
8.10.6	BUMP_SPECIFICATION_DRAWING
8.10.7	BUMP_ATTACHMENT_METHOD
8.11.4	MPD_MSL_LEVEL
8.11.5	MPD_PACKAGE_DRAWING
8.12.1	QUALITY
8.12.2	TEST
8.13.1	TEXT
8.14.1	PARSE

This standard shall be read in conjunction with EN 62258-1.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN and CENELEC shall not be held responsible for identifying any or all such patent rights.

The following dates were fixed:

– latest date by which the EN has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2012-03-29
– latest date by which the national standards conflicting with the EN have to be withdrawn	(dow)	2014-06-29

Annex ZA has been added by CENELEC.

Endorsement notice

The text of the International Standard IEC 62258-2:2011 was approved by CENELEC as a European Standard without any modification.

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE When an international publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

Publication	Year	Title	EN/HD	Year
IEC 61360-4	2005	Standard data element types with associated classification scheme for electric components - Part 4: IEC reference collection of standard data element types and component classes	EN 61360-4 + corr. December	2005 2005
IEC 62258-1	-	Semiconductor die products - Part 1: Procurement and use	EN 62258-1	-
ISO 6093	1985	Information processing - Representation of numerical values in character strings for information interchange	-	-
ISO 8601	2004	Data elements and interchange formats - Information interchange - Representation of dates and times	-	-
ISO 10303-21	2002	Industrial automation systems and integration - Product data representation and exchange - Part 21: Implementation methods: Clear text encoding of the exchange structure	-	-
IPC/JEDEC J-STD-033B	2007	Handling, Packing, Shipping and Use of Moisture/Reflow Sensitive Surface Mount Devices	-	-

CONTENTS

INTRODUCTION

This International Standard is based on the work carried out in the ESPRIT 4th Framework project GOODDIE which resulted in publication of the ES 59008 series of European specifications. Organisations that helped prepare this document include the ESPRIT ENCAST and ENCASIT projects, the Die Products Consortium, JEITA, JEDEC and ZVEI.

The structure of this International Standard as currently conceived is as follows:

Under main title: IEC 62258: Semiconductor die products

Part 1:	Procurement and use
Part 2:	Exchange data formats
Part 3:	Recommendations for good practice in handling, packing and storage (Technical report)
Part 4:	Questionnaire for die users and suppliers (Technical report)
Part 5:	Requirements for information concerning electrical simulation
Part 6:	Requirements for information concerning thermal simulation
Part 7:	XML schema for data exchange (Technical report)
Part 8:	EXPRESS model schema for data exchange (Technical report)

Further parts may be added as required.

SEMICONDUCTOR DIE PRODUCTS –

Part 2: Exchange data formats

1 Scope and object

This Part of IEC 62258 specifies the data formats that may be used for the exchange of data which is covered by other parts of the IEC 62258 series, as well as definitions of all parameters used according to the principles and methods of IEC 61360. It introduces a Device Data Exchange (DDX) format, with the prime goal of facilitating the transfer of adequate geometric data between die manufacturer and CAD/CAE user and formal information models that allow data exchange in other formats such as STEP physical file format, in accordance with ISO 10303-21, and XML. The data format has been kept intentionally flexible to permit usage beyond this initial scope.

It has been developed to facilitate the production, supply and use of semiconductor die products, including but not limited to:

- wafers,
- singulated bare die,
- die and wafers with attached connection structures,
- minimally or partially encapsulated die and wafers.

This standard reflects the DDX data format at version **1.3.0**

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 62258-1, *Semiconductor die products – Part 1: Procurement and use*

IEC 61360-4:2005, *Standard data element types with associated classification scheme for electric components – Part 4: IEC reference collection of standard data element types, component classes303-21*

ISO 8601:2004, *Data elements and interchange formats – Information interchange – Representation of dates and times*

ISO 6093:1985, *Information processing – Representation of numerical values in character strings for information interchange*

IPC/JEDEC J-STD-033B:2007, *Handling, Packing, Shipping and Use of Moisture/Reflow Sensitive Surface Mount Devices*

ISO 10303-21:2002*, Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 62258-1 apply.

4 Requirements

Specific reference for parameter variables is made to the IEC 61360 data element type (DET) codes, which are defined in Part 4 of IEC 61360.

5 Device Data eXchange format (DDX) file goals and usage

5.1 To facilitate the transferral of data by electronic media from the device vendor to the end-user for use within a CAD or CAE system, a data file format, **D**evice **D**ata e**X**change, (**DDX**), shall be used. This data file format has been deliberately kept flexible, to permit further enhancements and additions for future use.

5.2 It is strongly recommended that **D**evice **D**ata e**X**change files have the three letter **DDX** file extension, and a **D**evice **D**ata e**X**change file shall hereon be referred to as a **DDX** file.

5.3 Data that are to be transferred from a device vendor to a user shall be contained in a single computer-readable DDX file, and the minimum contents of this file shall suffice a geometric CAD/CAE software design system. The file shall be textually readable, to permit simple manual verification.

5.4 The DDX file and its data contents shall be independent of both computer machine and operating system.

5.5 The DDX file contents shall include mechanical and interconnectivity information, but may additionally include electrical and functional data.

5.6 The DDX file may contain data for one or more devices, and shall be capable of being used as a library file by a CAD/CAE software design system. The file may contain one or more sets of data for the same device type, each having different delivery forms, such as bumped die, bare die, and Chip-Scale packaging.

5.7 The DDX file shall be capable of being simply or automatically generated, such as by an ASCII text editor or a spreadsheet.

5.8 The DDX file shall be capable of referencing additional external files, such as simulation and thermal model files.

5.9 All data shall be defined in such a way that conversion to or from other exchange formats is possible, such as GDSII and CIF for geometric data of die. As close compatibility to the existing DIE (Die Information Exchange) data as possible is desired, to facilitate simple translation of partial DIE data files.

5.10 Definitions of parameters shall be in conformity with IEC 61360 (refer to Clause 5 of IEC 62258-1).

6 DDX file format and file format rules

NOTE 1 Version 1.2.1 of DDX supersedes version 1.0.0 contained in ES 59008-6-1.

NOTE 2 Version 1.3.0 of DDX supersedes version 1.2.1 contained in IEC 62258-2:2005.

Refer to Clause 1 for the DDX version of this standard.

6.1 Data validity

6.1.1 All data not complying with the data syntax (refer to 7.3) shall be treated as a remark and, as such, ignored.

6.1.2 All mandatory data shall be present. Missing data shall be flagged as an error, rendering that data unusable.

6.1.3 Mathematical operations, calculations or formulae shall not be permitted within numeric data.

6.2 Character set

6.2.1 The DDX file shall be an ASCII compatible text file with suitable line termination. Line termination will depend upon the operating system. DOS/Windows ® generally uses a carriage/line-feed <CR/LF> terminator (ASCII 0Dh/0Ah), whereas UNIX® invariably relies solely upon a line-feed <LF> (ASCII 0x0A) terminator, the carriage return <CR> (ASCII 0x0D) being present by implication.

6.2.2 ASCII characters 0x00 to 0x7F are permitted, ASCII characters 0x80 to 0xFF shall be ignored.

6.2.3 All text data shall be case independent.

6.2.4 Space characters (ASCII 20h) and tab characters (ASCII 09h) shall both be treated as space separators, multiple space and tab characters will syntactically be treated as a single space separator.

6.3 SYNTAX RULES

6.3.1 All data lines shall be terminated with a semicolon: “;”.

6.3.2 A comma “,” shall be used as a data separator.

6.3.3 Lines beginning with a hash “#” shall be treated as an intentional comment. All data on that line shall be ignored.

6.3.4 Underscores “_” shall be ignored in a variable or property name, and may be used as intermediate name separators. Underscores are valid within textual string and name data.

6.3.5 Braces are used to open and close structures or BLOCKs. An open brace “{“ shall be used to begin a structure or block, and a close brace “}” shall be used to terminate a structure or block.

6.3.6 Brackets “()” shall be permitted, then ignored, in numeric data for clarity (e.g. in co-ordinate pairs).

6.3.7 To accommodate typical spreadsheet CSV (Comma Separated Variable) format outputs, textual data may be inside double quotes “”, and matching pairs of double quotes shall be ignored.

6.3.8 There is no specific line continuation character. A textual string opened with a double quote ‘”’ shall close with a matching double quote ‘”’, irrespective of the number of line breaks within that text. As all DDX commands terminate with a semicolon, the non-textual data will be deemed to have ended at that semicolon. Textual data will be deemed to have ended at the semicolon following the closing double quote. Textual data not enclosed within double quotes may not include line break or control characters, and shall terminate at the first occurrence of a semicolon. Textual data following this semicolon will be treated as erroneous data and discarded.

6.3.9 For practicality, readability and ease of line parsing, is it recommended that the line length (between line termination characters) does not exceed 255 characters. It is further strongly recommended that a maximum limit of 1 023 characters per line be imposed to prevent other parsing software from having an input buffer overrun error.

7 DDX file content

7.1 DDX file content rules

7.1.1 Block structure

Data shall only exist within a block structure, referred to as a DEVICE block, and one or more DEVICE blocks, each containing data, may exist within a single file. Each DEVICE block is unique, and shall only contain data relevant to a single device, having a specific device form. All data within each DEVICE block shall be treated as being local and unique only to that block. (Refer to 6.3.4)

7.1.2 Parameter types

There are two types of parameters use for data, structures and variables, and these parameters shall only exist with a DEVICE block:

- a structure determines a set or multiple sets of data having different data types.
- a variable is equated to a single or multiple data of a single data type.

7.1.3 Data types

Data types are as follows.

7.1.3.1 Textual string data

All ASCII characters from ASCII 20h to ASCII 7Fh are permitted within textual data, characters including and above ASCII 80h shall be ignored. Consideration may be given to special print and display control characters to permit the printing of underscore or overscore characters. It is advised that textual string data is placed within pairs of double quotes, refer to 6.3.7.

7.1.3.2 Textual name data

All names shall be unique, and shall only consist of the following characters from the ASCII character set: -

A-Z a-z 0-9 $ - % & ! @ _ .

When textual name data are used to form a file name, it is advisable for the name to be limited to eight characters for the file name and to three characters for the file extension, with a point "**.**" used as the name/extension delimiter, in line with many common operating systems. It is advisable for textual name data to be placed within pairs of double quotes (refer to 6.3.7).

Note that all textual name data is case independent, and spaces are not permitted within a textual name.

7.1.3.3 Real numeric data

Real numeric data shall comply with ISO 6093:1985, and shall consist of the characters: -

0-9 + - . E e

The data values may be signed, and use engineering or scientific notation, but shall not include dimensional units, e.g.

90008, 9000.80, 9.0008E5, -5207, -5.207E3, 0.102, 102E-3

Note that a comma "**,**" is used as a data separator, and therefore shall not be used as a replacement for a decimal point "**.**".

7.1.3.4 Integer numeric data

Integer numeric data values shall comply to ISO 6093:1985, and only the characters **0** to **9** are permitted. Integers shall be unsigned, and shall not include dimensional units.

For practical purposes, an integer shall be limited to 16-bit resolution, i.e. integer values between and including 0 to 65536 only are acceptable.

7.1.3.5 Date data

Date data values shall comply with ISO 8601:2004 format, and may include time information as well, e.g.

"YYYY-MM-DD", "YYYYMMDD", "YYYY-MM-DDTHH:MM:SS".

7.1.4 Forward references

To permit single-pass parsing, no variable identifier or variable name shall be referenced prior to being defined.

7.1.5 Units

All units shall belong to the SI system, apart from the geometric unit of the micron (10^{-6}m), the inch and the mil (10^{-3} inch). Only one unit of dimension shall be permitted within a single **DEVICE** block. Note that the inch and the mil are non-preferred units, and are only present due to continued common usage.

7.1.6 Co-ordinate data

In all co-ordinate data, the **X** co-ordinate shall precede the **Y** co-ordinate and the **Y** co-ordinate shall precede the **Z** co-ordinate (i.e. **X,Y** or **X,Y,Z**).

The **X** co-ordinate shall be the horizontal axis (numerically left to right), the **Y** co-ordinate shall be the vertical axis (numerically bottom to top), and the **Z** co-ordinate shall be depth axis (numerically near to far).

7.1.7 Reserved words

All parameter names shall be considered as reserved and no variable identifier or variable name shall be permitted to have the same name. This restriction does not apply to free form textual data within quotes or double quotes (as 7.1.3.1 and 7.1.3.2).

7.2 DDX DEVICE block syntax

```
DEVICE device_name device_form {
        relevant die data ……
}
```

The **DDX** file may contain one or more **DEVICE** blocks, all data pertaining to a particular device shall be embedded within the relevant block. (Refer to clause 6.1.1 and clause 7.1.1).

A **DEVICE** block is opened by the **DEVICE** keyword and opening brace "**{**", (as shown), and the **DEVICE** block is closed by the matching closing brace "**}**".

Data not within a **DEVICE** block structure shall be treated as a remark, permitting the future addition of checksum information, file creation date and historical data etc., within the **DDX** file, without affecting the actual device data.

The **device_name** is the given name by which the device shall be referred, and the **device_form** is the mechanical form of the device to which the block data pertains.

Valid data for the **device_form** variable are:

- **bare_die,**
- **bumped_die,**
- **lead_frame_die**
- **minimally_packaged_device (**or **MPD).**

Further **device_form** types may be added at a later stage, refer to IEC 61360-4:2005, AAD004-001, "die type code", for further details.

Only one **DEVICE** block having **device_name** of type **device_form** shall be present within the **DDX** file, but duplication of either **device_name** or **device_form** is permissible.

An example of a typical **DDX** file arrangement of **DEVICE** blocks:-

```
DEVICE name1 bare_die {
relevant data for device "name1" as a bare die..
}
DEVICE name1 bumped_die {
relevant data for device "name1"as a bumped die..
}
DEVICE name2 mpd {
relevant data for device "name2" as a minimally packaged device..
}
DEVICE name2 bare_die {
relevant data for device "name2" as a bare die..
}
DEVICE name1 mpd {
relevant data for device "name1" as a minimally packaged device..
}
DEVICE name3 bare_die {
relevant data for device "name3" as a bare die..
}
```

In the above example, there are three occurrences of a **DEVICE** block for device "name1", and two occurrences of a **DEVICE** block for device "name2", but each of these **DEVICE** blocks specify a different **device_form**. The order or sequencing of the **DEVICE** blocks has no relevance.

7.3 DDX data syntax

Property = value [, value];

<**property**>[*equate separator*]<value/variable {*separator* <value/variable> }>[*data terminator*][*line terminator*]

<**property**>	::=	Parameter name
[*space*]	::=	{space character (20h) or tab character (09h)}0+
[*equate separator*]	::=	[space]{equal **=**}[space]
[*separator*]	::=	[space]{comma ,}[space]
[*data terminator*]	::=	[space]{semicolon;}
[*line terminator*]	::=	{CR or CR/LF}

For example:

```
thickness   =   100.0 ;
Thickness=470;
geometric_units=micron;
geometricunits     =     micron;
GeometricUnits= “millimetres”;
terminal_type = T1, Circle, 220;
Terminal_Type = T2, Rectangle,  200 ,  250;
TerminalType = T2, O, (200, 250);
TERMINALTYPE     =     T2, O,  200  ,  250;
```

Thus, **terminal_type, Terminal_Type**, **TerminalType** and **TERMINALTYPE** will all reference the same parameter name.

8 Definitions of DEVICE block parameters

8.0 General usage notes

8.0.1 Device form notes

Where a parameter is unique to the **device_form**, as defined in the **DEVICE** block, the parameter will be preceded with the following …

8.0.1.1 **DIE_** data parameter is unique to bare die or bumped die form

8.0.1.2 **BUMP_** data parameter is unique to only bumped die

8.0.1.3 **MPD_** data parameter is unique to a minimally packaged device, such as a CSP

8.0.1.4 **WAFER_** data parameter is unique to a die device delivered at wafer level

8.0.1.5 **LEAD_** data parameter is unique to a die device with attached lead frame.

8.0.2 Data Parameter Items

Within the following list of data parameters (see 8.1 onwards), the following items are shown:

8.0.2.1 the parameter name, as used syntactically within the **DDX** file,

8.0.2.2 the parameter type, indicating either a variable or structure data type,

8.0.2.3 the parameter function, determining its usage and meaning,

8.0.2.4 the parameter value, indicating the type of data expected,

8.0.2.5 any parameter limitation, indicating any limitation within the **DEVICE** block,

8.0.2.6 parameter dependencies, highlighting parameters that need to be declared prior to invocation,

8.0.2.7 one or more practical examples, and

8.0.2.8 any relevant notes.

A brief table of parameters is given in Annex F, and a working example of a full **DDX DEVICE** block is given in Annex A, with its expected graphical output in Annex C.

All parameters shall conform to the relevant IEC 61360 Data Element Type (DET) codes, as defined in IEC 61360-4:2005. Refer to Annex F for a cross-reference table.

8.0.3 Terms and conventions

A point of electrical connection is called a terminal. This may be a bond-pad for a bare die, and may equally refer to the landing or connection footprint area required by an interconnection medium. It is a common convention for die to have the initial terminal, numbered 1, in the upper left hand corner of the die, and for terminal or pin numbering to continue counter-clockwise in sequence.

The X co-ordinate dimensions are for the length in the horizontal plane with increasing positive values to the right. The Y co-ordinate dimensions are for the width in the horizontal plane with increasing positive values away from the user’s view. The Z co-ordinate dimensions are for height in the vertical direction with increasing positive values upwards (towards the viewer).

As a point of reference, all die components, including bumped die, are generally viewed from above with the active side upwards.

8.0.4 Summary of general rules

8.0.4.1 All valid data shall be contained within a **DEVICE** block (refer to 7.1.1).

8.0.4.2 Any local or unique parameter, such as a name, shall be defined prior to its usage.

8.0.4.3 All parameters shall conform to the relevant IEC 61360 DET codes, as defined in Part 4 of IEC 61360 (refer to 5.8 and Annex F).

8.0.4.4 The units of measurement, **GEOMETRIC_UNITS**, shall be defined before any geometric variable is defined.

8.0.4.5 The geometric origin, **GEOMETRIC_ORIGIN**, and the geometric view, **GEOMETRIC_VIEW**, shall be defined before any geometric co-ordinates are defined.

8.0.4.6 The **TERMINAL_COUNT** parameter shall be defined before any **TERMINAL** parameters are referred to and the number of **TERMINAL** parameters shall not exceed the **TERMINAL_COUNT** value.

8.0.4.7 The **TERMINAL_TYPE_COUNT** parameter shall be defined before any **TERMINAL_TYPE** parameters are referred to and the number of **TERMINAL_TYPE** parameters shall not exceed the **TERMINAL_TYPE_COUNT** value.

8.1 BLOCK DATA

8.1.1 DEVICE_NAME Parameter

This is defined within the **DEVICE** block heading as the **device_name** parameter

Parameter Name	**device_name**
Parameter Type	Variable, refer to 7.2 on **DEVICE** blocks
Parameter Function	Defines the device manufacturer's type number or the reference name.
Parameter Values	Textual name data, as 7.1.3.2
Example	`SN74LS04`
Reference	See 7.1.7, 7.2 and Annex G.

8.1.2 DEVICE_FORM Parameter

This is defined within the **DEVICE** block heading by the **device_form** parameter.

Parameter Name	**device_form**
Parameter Type	Variable, refer to 7.2 on **DEVICE** blocks
Parameter Function	Defines the physical form of the device.
Parameter Values	Textual string data, as 7.1.3.1
Example	`bare_die, bumped_die, MPD`
Reference	Subclause 7.2 and Annex G.

8.1.3 BLOCK_VERSION Parameter

Parameter Name	**BLOCK_VERSION**
Parameter Type	Variable
Parameter Function	Specifies the version number and/or issue number of the DEVICE block.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`BLOCK_VERSION = “1.0A”;`
Reference	Annex G.

8.1.4 BLOCK_CREATION_DATE Parameter

Parameter Name	**BLOCK_CREATION_DATE**
Parameter Type	Variable
Parameter Function	Specifies the date that the DEVICE block was created and last edited.
Parameter Values	Date, as 7.1.3.5
Limitations	Shall be declared only once within a single DEVICE block.
Example	`BLOCK_CREATION_DATE = “1997-12-25”;`
Reference	Annex G.

8.1.5 VERSION Parameter

Parameter Name	**VERSION**
Parameter Type	Variable
Parameter Function	Specifies the revision/version number of the DDX standard, (currently at version 1.3.0), to which this DEVICE block conforms.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`VERSION = “1.3.0”;`
Notes	Refer to Clause 1 for the version of this standard document. Note that this document may not be the latest version, so refer to your Standards Authority if in doubt.
Reference	Annex G.

8.2 DEVICE DATA

8.2.1 DIE_NAME Parameter

Parameter Name	**DIE_NAME**
Parameter Type	Variable
Parameter Function	Specifies the name of the die or mask set from which the die was produced. This may be different to the DEVICE_NAME parameter.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`DIE_NAME = “XXC345”;`
Reference	Annex G.

8.2.2 DIE_PACKAGED_PART_NAME Parameter

Parameter Name	**DIE_PACKAGED_PART_NAME**

Parameter Type	Variable
Parameter Function	Specifies the manufacturers part name for the equivalent packaged part, where available or applicable. This may be different to the DEVICE_NAME parameter.
Parameter Values	Textual string data, as 7.1.3.1
Example	`DIE_PACKAGED_PART_NAME = “SN5405JN”;`
Notes	Used to reference the identical packaged die part, not merely similar function, when supplied by the same manufacturer in packaged form.

8.2.3 DIE_MASK_REVISION Parameter

Parameter Name	**DIE_MASK_REVISION**
Parameter Type	Variable
Parameter Function	Specifies the mask revision details associated with that particular version on the die.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DIE_MASK_REVISION = “RTDAC1”;`
	`DIE_MASK_REVISION = “9033-2-101-M5/2”;`

Notes	Intended to ensure that the die data matches the geometric version of the actual die.
Reference	Annex G.

8.2.4 MANUFACTURER Parameter

Parameter Name	**MANUFACTURER**
Parameter Type	Variable
Parameter Function	Specifies the manufacturer or fabrication house of the device.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`MANUFACTURER = “Fuzziwuz Logic Inc.”;`

8.2.5 DATA_SOURCE Parameter

Parameter Name	**DATA_SOURCE**
Parameter Type	Variable
Parameter Function	Specifies the source of the device data, essentially where this is different from the manufacturer or fabrication house.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DATA_SOURCE = “AnyChip Technology Ltd.”;`
	`DATA_SOURCE = “Good-Die database”;`

8.2.6 DATA_VERSION Parameter

Parameter Name	**DATA_VERSION**
Parameter Type	Variable
Parameter Function	Specifies the revision of the data source use to determine the parameters within the DEVICE block. This parameter is linked to 8.2.5, the DATA_SOURCE parameter.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`DATA_VERSION = “Initial Issue 1.0”;`
Reference	Refer to Annex G.

8.2.7 FUNCTION Parameter

Parameter Name	**FUNCTION**
Parameter Type	Variable
Parameter Function	A brief description of the devices’ function.
Parameter Values	Textual string data, as 7.1.3.1

Limitations	Shall be declared only once within a single DEVICE block.
Example	`FUNCTION = “16 Bit Microprocessor”;`
Notes	IEC 61360-4:2005 specifies certain functional and application classes that may be used.

8.2.8 IC_TECHNOLOGY Parameter

Parameter Name	**IC_TECHNOLOGY**
Parameter Type	Variable
Parameter Function	Specifies the fabrication technology of the device.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`IC_TECHNOLOGY = “CMOS”;`
	`IC_TECHNOLOGY = “bipolar”;`
	`IC_TECHNOLOGY = “bicmos”;`
	`IC_TECHNOLOGY = “GaAs”;`

8.2.9 DEVICE_PICTURE_FILE Parameter

Parameter Name	**DEVICE_PICTURE_FILE**
Parameter Type	Variable
Parameter Function	Specifies the name(s) of the graphics or picture file(s) representing the device.
Parameter Values	Textual name data, as 7.1.3.2

Example	`DEVICE_PICTURE_FILE = “DIE001.JPG”;`
	`DEVICE_PICTURE_FILE = “AA0B0C.SF”, “FRED.GIF”;`

Notes	The picture file may be an actual photograph, or a pictorial representation of the device. The file extent should indicate the file format used. Only file names, without relative or absolute path names, shall be used. Multiple parameters (picture file names) may be introduced within a single declaration, or multiple declarations maybe be employed to the same effect. There is no scaling or positional data requirement.
Reference	Annex I, Notes 3 and 4.

8.2.10 DEVICE_DATA_FILE Parameter

Parameter Name	**DEVICE_DATA_FILE**
Parameter Type	Variable
Parameter Function	Specifies the name(s) of a file(s) containing pertinent data, such as technical specifications, procurement documents or general data-sheets.
Parameter Values	Textual name data, as 7.1.3.2

Example	`DEVICE_DATA_FILE = “SpecSheet.PDF”;`
	`DEVICE_DATA_FILE = “AA0B0C.DOC”, “AAOBOC.TXT”;`

Notes	The data file may be of any recognised format. The file extent should indicate the file format used. Only file names, without relative or absolute path names, shall be used. Multiple parameters (data file names) may be introduced within a single declaration, or multiple declarations maybe be employed to the same effect.
Reference	Annex I, Notes 3 and 4.

8.3 GEOMETRIC DATA

8.3.1 GEOMETRIC_UNITS Parameter

Parameter Name	**GEOMETRIC_UNITS**
Parameter Type	Variable
Parameter Function	Specifies the geometric units that shall apply to all geometric values within the DEVICE block.
Parameter Values	Textual string data, as 7.1.3.1. One of the following values: ▪ micrometre, or micron, ▪ metre, ▪ millimetre, ▪ inch ▪ mil (1.0E-3 inch)
Limitations	Shall be declared only once within a single DEVICE block.

Example	`GEOMETRIC_UNITS = microns;`
	`GEOMETRIC_UNITS = mil;`

Notes	This GEOMETRIC_UNITS parameter shall be declared before any geometric units are used. The micron is generally the default dimensional unit for die dimensions, and the inch and mil are non-preferred units.
Reference	Subclause 7.1.5 and Annex E.

8.3.2 GEOMETRIC_VIEW Parameter

Parameter Name	**GEOMETRIC_VIEW**
Parameter Type	Variable
Parameter Function	Specifies the geometric view that shall apply to all geometric shapes within the DEVICE block.
Parameter Values	Textual string data, as 7.1.3.1. One of the following values: • **TOP** meaning active side upwards, and • **BOTTOM** meaning active side downwards.
Limitations	Shall be declared only once within a single DEVICE block.

Example	`GEOMETRIC_VIEW = top;`
	`GEOMETRIC_VIEW = “bottom”;`

Notes	The GEOMETRIC_VIEW parameter shall be declared before any geometric shapes are created. It would be common for a bare die and packaged part to be viewed in the “TOP” view, whereas bumped die may well be viewed from the “BOTTOM” (i.e. through the substrate).
Reference	Annex E.

8.3.3 GEOMETRIC_ORIGIN Parameter

Parameter Name	**GEOMETRIC_ORIGIN**
Parameter Type	Variable
Parameter Function	Determines the X- and Y-geometric origin from which all other co-ordinate pairs are referenced. The origin is given with respect to the geometric centre of the die in the units specified by the GEOMETRIC_UNITS parameter.
Parameter Values	Real X co-ordinate origin, Real Y co-ordinate origin, as 7.1.3.3
Dependencies	GEOMETRIC_UNITS, SIZE
Limitations	Shall be declared only once within a single DEVICE block.
Example	`GEOMETRIC_ORIGIN = -6000,-7500;`
Notes	Dimension values are in GEOMETRIC_UNITS. The origin co-ordinate pair values relate to the geometric die centre, refer to Figure 1. for further explanation.
Reference	Annex E.

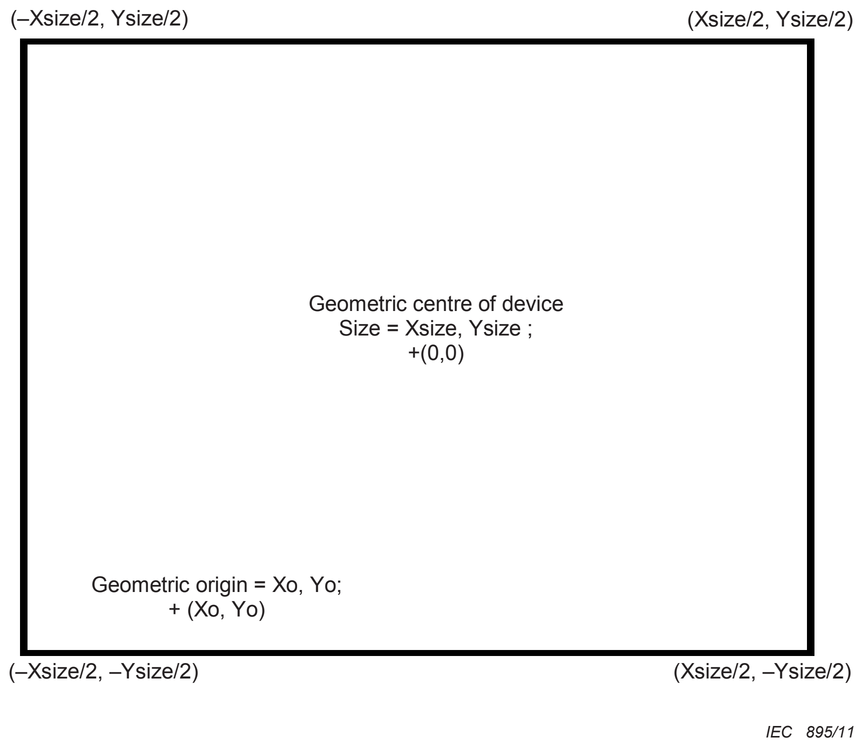


Figure 1 – Relationship between geometric centre and geometric origin

The GEOMETRIC_ORIGIN is treated as an offset for all co-ordinate data, so that the GEOMETRIC_ORIGIN values are ***added*** to ***all*** individual co-ordinate data pairs to give the X- and Y- position relative to the geometric centre of the device.

The primary use of the GEOMETRIC_ORIGIN parameter is to permit the centring of the origin on a terminal or other geometric feature, rather than an arbitrary geometric position, as, in practice, SIZE may be subject to an asymmetric tolerance so that all related references to this geometric centre could therefore also be subject to a tolerance error.

8.3.4 SIZE Parameter

Parameter Name	**SIZE**
Parameter Type	Variable
Parameter Function	Determines the X- and Y- dimensions of the device, and optionally specifies its shape as an ellipse.
Parameter Values	Real X-dimension, Real Y-dimension, (as 7.1.3.3), {Ellipse}
Dependencies	GEOMETRIC_UNITS, GEOMETRIC_VIEW
Limitations	Shall be declared only once within a single DEVICE block.
Example	`SIZE = 250, 500.5;` `SIZE = 350, 350, E;`
Notes	Dimension values are in GEOMETRIC_UNITS. In the latter example, the character “E” as the 3rd parameter defines an ellipse, in this instance a circular die of 350 GEOMETRIC_UNITS in diameter.
Reference	Annex E.

8.3.5 SIZE_TOLERANCE Parameter

Parameter Name	**SIZE_TOLERANCE**
Parameter Type	Variable
Parameter Function	Specify the geometric tolerance(s) of the SIZE parameter values.
Parameter Values	Real in GEOMETRIC_UNITS, as 7.1.3.3

Dependencies	GEOMETRIC_UNITS, SIZE, GEOMETRIC_VIEW
Limitations	Shall be declared only once within a single DEVICE block.

Example	`SIZE_TOLERANCE = 0.5;`
	`SIZE_TOLERANCE = -0.2,0.5;`
	`SIZE_TOLERANCE = -0.2,0.5,-0.1,0.4;`

Notes	One, two or four values may be given:- Where a single tolerance value is given, the tolerance shall be taken as an unsigned ± value for both the X- and Y-axis. Where two tolerance values are given:- the first value shall be taken as the minimum for both the X-axis and the Y-axis, the second value shall be taken as the maximum for both X-axis and the Y-axis. Where four tolerance values are given:- the first value shall be taken as the minimum for the X-axis, the second value shall be taken as the maximum for the X-axis, the third value shall be taken as the minimum for the Y-axis, and the fourth value shall be taken as the maximum for the Y-axis.

8.3.6 THICKNESS Parameter

Parameter Name	**THICKNESS**
Parameter Type	Variable
Parameter Function	Determines the thickness (Z-dimension) of the device.
Parameter Values	Real Z-dimension value, as 7.1.3.3
Dependencies	GEOMETRIC_UNITS
Limitations	Shall be declared only once within a single DEVICE block.
Example	`THICKNESS = 10.5;`
Notes	Dimension values are in GEOMETRIC_UNITS.

8.3.7 THICKNESS_TOLERANCE Parameter

Parameter Name	**THICKNESS_TOLERANCE**
Parameter Type	Variable
Parameter Function	Specifies the tolerance(s) of the THICKNESS parameter that may be expected due to normal process variations.
Parameter Values	Real in GEOMETRIC_UNITS, as 7.1.3.3
Dependencies	GEOMETRIC_UNITS, THICKNESS
Limitations	Shall be declared only once within a single DEVICE block.

Example	`THICKNESS_TOLERANCE = 2.5;`
	`THICKNESS_TOLERANCE = -1.2,1.5;`

Notes	One or two values may be given:- Where a single tolerance value is given, the tolerance shall be taken as an unsigned ± value. Where two tolerance values are given:- the first value shall be taken as the minimum, and the second value shall be taken as the maximum.

8.3.8 FIDUCIAL_TYPE Parameter

Parameter Name	**FIDUCIAL_TYPE**
Parameter Type	Structure
Parameter Function	The FIDUCIAL_TYPE structure assigns a fiducial type name and defines the associated graphic file and size of an individual fiducial type. The structure may be used to define a single fiducial type, or a multiple of fiducial types.
Dependencies	GEOMETRIC_UNITS, GEOMETRIC_VIEW
Notes	A fiducial only exists as a graphic shape within a rectangle, the graphic data for the fiducial is held within an external graphic file. The co-

	ordinate pairs for this rectangle are relative only to the fiducial shape type, and are used as references when placing the shape.
Reference	Annexes E and I.

Single FIDUCIAL_TYPE definition syntax:

FIDUCIAL_TYPE *Fiducial_type_name* = *Fiducial_file_name*, *X-size, Y-size;*
FIDUCIAL_TYPE *Fiducial_type_name* = *Fiducial_file_name*, *X-size, Y-size;*

Multiple FIDUCIAL_TYPE definition syntax:

```
FIDUCIAL_TYPE {
        Fiducial_type_name = Fiducial_file_name, X-size, Y-size;
        Fiducial_type_name = Fiducial_file_name, X-size, Y-size;
        Fiducial_type_name = Fiducial_file_name, X-size, Y-size;
}
```

where:

8.3.8.1 Fiducial_type_name

Textual reference name (as 7.1.3.2) for a fiducial type, which shall be unique within the DEVICE block.

8.3.8.2 Fiducial_file_name

This is the name of the file (as 7.1.3.2) that holds the fiducial as graphic data. The graphic data type shall be indicated by the file extension code, such as “BMP”, “GIF”, “DXF” etc. The data type is not specified with this standard, and it is up to the CAD/CAM software as to what can and cannot be displayed. The graphic shall be treated as being contained within a rectangle of X-size, Y-size dimensions.

8.3.8.3 X-size, Y-size

These real numeric parameters (as 7.1.3.3) comprise a co-ordinate pair, and determine the rectangle size of the fiducial graphic. The geometric centre of the fiducial graphic shall be determined as being (X-size/2, Y-size/2).

Example 1

```
FIDUCIAL_TYPE Fid1 = “tile.bmp”, 200, 250;
```

This example describes a fiducial, found as an external file “`tile.bmp`”, uniquely referred to as `Fid1`, with an X-size of 200 units, and a Y-size of 250 units. The units are defined by the GEOMETRIC_UNITS parameter.

Example 2

```
FIDUCIAL_TYPE  {
      fidu1 = “graphic1.bmp”, 200, 300;
      fidu2 = “graphic2.dxf”, 500, 500;
      fidu3 = “graphic3.gif”, 100, 100;
}
```

This multiple definition example describes three separate fiducial types and shapes:

- `fidu1` is contained in file “`graphic1.bmp`”, of size 200 by 300 units.
- `fidu2` is contained in file “`graphic2.dxf`”, of size 500 by 500 units.
- `fidu3` is contained in file “`graphic3.gif`”, of size 100 by 100 units.

8.3.9 FIDUCIAL Parameter

Parameter Name	**FIDUCIAL**
Parameter Type	Structure
Parameter Function	The FIDUCIAL structure defines the location and orientation of each individual graphic fiducial. As a structure, FIDUCIAL may be used to define a single fiducial, or a multiple of fiducials.
Dependencies	GEOMETRIC_VIEW, FIDUCIAL_TYPE, GEOMETRIC_UNITS, and GEOMETRIC_ORIGIN.
Notes	Each fiducial type name used shall have been previously declared in a FIDUCIAL_TYPE invocation, and co-ordinate pair information shall relate to the GEOMETRIC_ORIGIN of the device.
Reference	Annex E.

Single **FIDUCIAL** definition syntax:

> **FIDUCIAL F_*n*** = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
> **FIDUCIAL F_*n*** = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*

Multiple FIDUCIAL definition syntax:

```
FIDUCIAL {
        F_n = Fiducial_type_name, X-co., Y-co., orientation;
        F_n = Fiducial_type_name, X-co., Y-co., orientation;
        F_n = Fiducial_type_name, X-co., Y-co., orientation;
}
```

where:

8.3.9.1 F_*n*

Unique fiducial identifier, where "***n***" is the actual fiducial number (Integer, as in 7.1.3.4), numbering from top left anti-clockwise. Note that the underscore character is optional, it is used for clarity only.

8.3.9.2 Fiducial_type_name

Textual name (as in 7.1.3.2) of a referenced FIDUCIAL_TYPE shape (as 8.3.8.1), which shall have been previously declared, (refer to 7.1.4).

8.3.9.3 X-co

Numeric real X co-ordinate value (as 7.1.3.3) for location of the centre of the fiducial shape, in units defined by the GEOMETRIC_UNITS parameter. This value is relative to the X co-ordinate value of the GEOMETRIC_ORIGIN, and this parameter is identical in operation to that described in 8.4.5.4.

8.3.9.4 Y-co

Numeric real Y co-ordinate value (as 7.1.3.3) for location of the centre of the fiducial shape, in units defined by the GEOMETRIC_UNITS parameter. This value is relative to the Y co-ordinate value of the GEOMETRIC_ORIGIN, and this parameter is identical in operation to that described in 8.4.5.5.

8.3.9.5 Orientation

Orientation value, of integer values from 0 to 360 (see 7.1.3.4). This is the angle of clockwise rotation in degrees about the geometric reference centre of the fiducial shape. If the letters "**MX**" are included, then the orientation of the fiducial shape shall be mirrored in the X-axis, similarly if the letters "**MY**" are included, then the orientation of the fiducial shape shall be mirrored in the Y-axis. Both "**MX**" and "**MY**" may be present simultaneously (refer to Annex D for a graphical representation), and all mirroring shall be done about the geometric reference centre of the fiducial shape. Note that the mirroring operation shall be carried out first, then the fiducial shall be rotated by the orientation angle. This parameter is identical in operation to that described in 8.4.5.6

Example 1

```
FIDUCIAL Fid007 = fidu1, 5000, 7000, MX0;
```

This example declares that fiducial **Fid007**......

- is located at X = 5000 units, Y = 7000 units,
- is a FIDUCIAL_TYPE named "fidu1", mirrored on the X-axis and orientated at 0^o.

Example 2

```
FIDUCIAL F_18 = fiduX1, 1000, 2200, 90;
```

This example declares that fiducial **F_18** .....

- is located at X = 1000 units, Y = 2200 units
- is a FIDUCIAL_TYPE named "fiduX1", orientated (rotated) by 90^o clockwise.

Example 3

```
FIDUCIAL {
  Fid007-fidu1,  5000, 7000, MX0;
  F_18 = fiduX1, 1000, 2200, 90;
}
```

This multiple fiducial definition example declares identical fiducial data to that shown in Examples 1 and 2.

8.4 TERMINAL DATA

8.4.1 TERMINAL_COUNT Parameter

Parameter Name	**TERMINAL_COUNT**
Parameter Type	Variable
Parameter Function	Specifies the number of electrical terminal or points of connection.
Parameter Values	Integer, as in 7.1.3.4
Limitations	Shall be declared only once within a single DEVICE block.
Example	`TERMINAL_COUNT = 44;`
Notes	The TERMINAL_COUNT value shall be declared before any TERMINAL declaration or usage.

8.4.2 TERMINAL_TYPE_COUNT Parameter

Parameter Name	**TERMINAL_TYPE_COUNT**
Parameter Type	Variable
Parameter Function	Specifies the number of different connection or bond terminal types or shapes.
Parameter Values	Integer, as in 7.1.3.4
Limitations	Shall be declared only once within a single DEVICE block.
Example	`TERMINAL_TYPE_COUNT = 4;`
Notes	The TERMINAL_TYPE_COUNT value shall be declared before any TERMINAL_TYPE declaration or usage.

8.4.3 CONNECTION_COUNT Parameter

Parameter Name	**CONNECTION_COUNT**
Parameter Type	Variable, optional

Parameter Function	Specifies the maximum number of connections used by the TERMINAL parameter. This parameter actually specifies the highest value for the connection ***conn_N*** numbers used in the TERMINAL parameter declaration (refer to 8.4.5.2).
Parameter Values	Integer, as in 7.1.3.4
Limitations	Shall be declared only once within a single DEVICE block.
Example	`CONNECTION_COUNT = 4;`
Notes	The CONNECTION_COUNT value should be declared before any TERMINAL declaration or usage. It shall be used as a limit check for the connection ***conn_N*** numbers. If the CONNECTION_COUNT parameter is not declared, then no check will take place.

8.4.4 TERMINAL_TYPE Parameter

Parameter Name	**TERMINAL_TYPE**
Parameter Type	Structure
Parameter Function	The TERMINAL_TYPE structure assigns a type name and defines the shape and size of an individual terminal type. As a structure, TERMINAL_TYPE may be used to define a single terminal type, or a multiple of terminal types.
Dependencies	GEOMETRIC_UNITS, GEOMETRIC_VIEW, TERMINAL_TYPE_COUNT
Notes	The co-ordinate pairs are relative only to the terminal shape type, and are used as references when placing the shape.
Reference	Annex E.

Single TERMINAL_TYPE definition syntax:

TERMINAL_TYPE *Terminal_type_name* = *Terminal_shape_type*, *Co-ordinates*.,,,;
TERMINAL_TYPE *Terminal_type_name* = *Terminal_shape_type*, *Co-ordinates*.,,,;

Multiple TERMINAL_TYPE definition syntax:

TERMINAL_TYPE {
Terminal_type_name = *Terminal_shape_type*, *Co-ordinates* .,,,;
Terminal_type_name = *Terminal_shape_type*, *Co-ordinates* .,,,;
Terminal_type_name = *Terminal_shape_type*, *Co-ordinates* .,,,;
}

where:

8.4.4.1 Terminal_type_name

Textual reference name for a terminal type, as in 7.1.3.2, which shall be unique within the DEVICE block.

8.4.4.2 Terminal_shape_type

Determines the terminal shape type (only the first letter need be used):

Table 1 – Terminal shape types

Char	DET data	Shape
R	RECT	**R**ectangle,
C	CIRC	**C**ircle,
E	ELL	**E**llipse,
P	POLY	**P**olygon.

8.4.4.3 Co-ordinates

Real co-ordinate values (refer to Table 2), as in 7.1.3.3.

This specifies the relative co-ordinates for the terminal shape, and in doing so, determines the geometric reference centre for the shape. The geometric reference centre of the shape will be used by the TERMINAL co-ordinates (see 8.4.5.4 and 8.4.5.5) to place the shape, and all rotation and mirroring will be about this geometric reference centre. Only the polygon shape can have a geometric reference centre other than the natural "centre-of gravity".

Table 2 – Terminal shape co-ordinates

Shape	Co-ordinates	Geometric reference centre
Rectangle	X-size, Y-size	The geometric centre (0,0) will occur at X-size/2:Y-size/2
Circle	Diameter	The geometric centre (0,0) will occur at diameter/2:diameter/2
Ellipse	X-axis, Y-axis	The geometric centre (0,0) will occur at X-axis/2, Y-axis /2
Polygon	X-co$_1$, Y-co$_1$,...... X-co$_N$, Y-co$_N$	An "N" sided polygon will have N pairs of co-ordinates, geometrically centred upon (0,0), with the assumption that the polygon shape will be completed with the vector (Xco$_N$,Yco$_N$ - Xco$_1$,Yco$_1$)

Example 1

```
TERMINAL_TYPE ShapeR1 = Rectangle,  200, 250;
```

This example describes a rectangle, uniquely referred to as ShapeR1, with an X-axis dimension of 200 units, and a Y-axis dimension of 250 units. The units are defined by the GEOMETRIC_UNITS parameter.

Example 2

```
TERMINAL_TYPE  {
      ShapeR1 = R, 200, 300;
      ShapeC2 = C, 250;
      ShapeE3 = E, 400, 200;
      ShapeP4 = P, (150, 200),(-150, 200),(-150,-150),(150,-150);
}
```

This example describes four separate terminal types and shapes:

- ShapeR1 is a rectangular terminal having an X-axis dimension of 200 units and a Y-axis dimension of 300 units,
- ShapeC2 is a circular terminal of 250 units in diameter,
- ShapeE3 is an elliptical terminal with the X-axis diameter of 400 units and Y-axis diameter of 200 units, and
- ShapeP4 is a 4-sided polygonal terminal shape, with a geometric reference centre of (0,0).

Note that only the first letter of the *Terminal_shape_type* descriptor is used, and that the *Terminal_type_names* are unique.

8.4.5 TERMINAL Parameter

Parameter Name	**TERMINAL**
Parameter Type	Structure

Parameter Function	The TERMINAL structure defines the name, location, orientation and electrical function of each individual connection terminal. As a structure, TERMINAL may be used to define a single terminal, or a multiple of terminals.
Dependencies	TERMINAL_COUNT, CONNECTION_COUNT, GEOMETRIC_VIEW, TERMINAL_TYPE, GEOMETRIC_UNITS, GEOMETRIC_ORIGIN.
Notes	Each terminal type name used shall have been previously declared in a TERMINAL_TYPE invocation, and co-ordinate pair information shall relate to the GEOMETRIC_ORIGIN of the device. Refer to Annex E.

Single TERMINAL definition syntax:

```
TERMINAL T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
               Terminal_name, IO_type;
TERMINAL T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
               Terminal_name, IO_type;
```

Multiple TERMINAL definition syntax:

```
TERMINAL {
          T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
               Terminal_name, IO_type;
          T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
               Terminal_name, IO_type;
          T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
               Terminal_name, IO_type;
         }
```

where:

8.4.5.1 T_*n*

Unique terminal identifier, where "***n***" is the actual terminal number (Integer, as 7.1.3.4), numbering from top left anti-clockwise. Note that the underscore character is optional, it is used for clarity only.

8.4.5.2 conn_N

Connection number, non-unique integer, as 7.1.3.4. This connection number is merely a place reference, and may optionally refer to a package pin; need not be present, or may be zero, 0, or left blank if unknown. The singular advantage of this connection number is in specifying and identifying multiple terminals that need to be connected together. The value of this connection number should be checked to ensure that it does not exceed the value determined by CONNECTION_COUNT, when specified.

8.4.5.3 Terminal_type_name

Textual name, as 7.1.3.2, of a referenced TERMINAL_TYPE terminal shape (as 8.4.3.1), which shall have been previously declared, (refer to 7.1.4).

8.4.5.4 X-co

Numeric real X co-ordinate value, as 7.1.3.3, for location of centre of the terminal shape, in units defined by the GEOMETRIC_UNITS parameter. This value is relative to the X co-ordinate value of the GEOMETRIC_ORIGIN.

8.4.5.5 Y-co

Numeric real Y co-ordinate value, as 7.1.3.3, for location of centre of the terminal shape, in units defined by the GEOMETRIC_UNITS parameter. This value is relative to the Y co-ordinate value of the GEOMETRIC_ORIGIN.

8.4.5.6 Orient

Orientation value, of integer values from 0 to 360, as 7.1.3.4. This is the angle of clockwise rotation in degrees about the geometric reference centre of the terminal shape. If the letters "**MX**" are included, then the orientation of the terminal shape shall be mirrored in the X-axis, similarly if the letters "**MY**" are included, then the orientation of the terminal shape shall be mirrored in the Y-axis. Both "**MX**" and "**MY**" may be present simultaneously (refer to Annex D for a graphical representation), and all mirroring shall be done about the geometric reference centre of the terminal shape. Note that the mirroring operation shall be carried out first, then the terminal shall be rotated by the orientation angle.

8.4.5.7 Terminal_name

Textual name, non-unique, as 7.1.3.2. This terminal name is for user reference and graphic display only, and may be omitted.

8.4.5.8 IO_type

A single letter indicating the terminal pin function type as given in Table 3, not mandatory. Further characters may be added as required.

Table 3 – Terminal IO types

Letter	Terminal Function
I	**I**nput, digital
O	**O**utput, digital
B	**B**i-directional, digital
G	**G**round connection
V	Supply, may be any supply type
A	**A**nalog pin, input or output
N	**N**o-connect pin, leave disconnected
U	**U**ndetermined, user programmable I/O
T	**T**est pin, connect only under manufacturers advice
X	Internally connected, do not connect.
H	Digital functional input pin, to be held at a logic **H**igh
L	Digital functional input pin, to be held at a logic **L**ow

T_*n* (see 8.4.5.1) shall always be unique, whereas the pin connection number (see 8.4.5.2), ***conn_N***, and terminal name (see 8.4.5.7), ***Terminal_name***, need not be unique.

Example 1

```
    TERMINAL Pin007 = 9, ShapeR1, (5000, 7000), MX0, VCC1, V;
```

This example declares that terminal "**Pin007**"......

- is connected to arbitrary connection # 9,
- is located at x = 5000 units, Y = 7000 units,

- is a TERMINAL_TYPE named “ShapeR1”, mirrored on the X-axis, orientated at 0°, and
- is called “VCC1”, and
- is a power supply pin.

Example 2

```
TERMINAL Conn08 = 17, ShapeP2, 5000, 7300, 90, qd2i, I;
TERMINAL Conn09 = 17, ShapeP2, 5000, 7800, 90, qd2o, O;
```

These example declare that terminals “**Conn08”** and “**Conn09”**.....

- are both connected to arbitrary connection # 17,
- are located at X = 5000 units, Y = 7300 units & X = 5000 units, Y = 7800 units respectively,
- are both a TERMINAL_TYPE named “ShapeP2”, orientated (rotated) by 90° clockwise,
- are named “qd2i” & “qd2o”
- and constitute a digital I/O bi-directional connection, with “qd2I” being an input (I) and “qd2o” being an output (O).

Example 3

```
TERMINAL Term10 = , ShapeP2, 5000, 7600, 0, , X;
```

This example declares that terminal “**Term10”**.....

- is unconnected (or with no specific connection reference),
- is located at X = 5000 units, Y = 7600 units,
- is a TERMINAL_TYPE named “ShapeP2”, orientated at 0°, and
- has no given name and
- is specified as not to be bonded to.

Example 4

```
TERMINAL T_44 = , ShapeR2, 25000, 97000, MXMY90, ,;
```

This example declares that terminal **T_44**.....

- is unconnected (or with no specific connection reference),
- is located at (25000:97000),
- is a TERMINAL_TYPE named “ShapeR2”, mirrored in both the X- and Y-axis, then orientated at (rotated clockwise by) 90°,
- has no given name, and
- has no specified electrical connection properties.

Example 5

```
TERMINAL {
      Pin007 = 9, ShapeR1, (5000, 7000), MX0, VCC1, V;
      Conn08 = 17, ShapeP2, (5000, 7300), 90, qd2i, I;
      Conn09 = 17, ShapeP2, (5000, 7800), 90, qd2o, O;
      Term10 = , ShapeP2, (5000, 7600), 0, , X;
      T_44 = , ShapeR2, (25000, 97000), MXMY90, ,;
}
```

This multiple terminal definition example declares identical terminal data to that shown in Examples 1 to 4.

8.4.6 TERMINAL_GROUP Parameter

Parameter Name	TERMINAL_GROUP
Parameter Type	Structure
Parameter Function	The TERMINAL_GROUP structure assigns a group of terminals to a single terminal-group identifier that may be used in place of a list of

	terminals. The structure may be used to define a single group or a multiple of groups. Refer to TERMINAL_GROUP Rules (voir 8.4.6.1) for detail.
Parameter Values	Two or more terminal or group identifiers, as defined in 8.4.5.1 and 8.4.6.3
Dependencies	TERMINAL (see 8.4.5)
Limitations	A TERMINAL_GROUP identifier must be unique, and may only referenced after it has been declared.
Reference	Annex B.

8.4.6.1 TERMINAL_GROUP Rules

8.4.6.1.1 Content

The elements forming a group may be terminal identifiers or group identifiers or any mixture of the two. A group must contain at least two elements.

8.4.6.1.2 Uniqueness

All elements within a group must be unique. A group may not contain groups that refer directly or indirectly to the same element.

8.4.6.1.3 Ordering

The order of elements within groups may be important. Where two or more groups are to be related in a permutation, then the elements within those groups must correspond across the groups.

8.4.6.1.4 Recursion

A group may not contain itself nor may it contain any group that refers directly or indirectly to itself

Single TERMINAL_GROUP definition syntax:

TERMINAL_GROUP **G_*n*** = _ *Terminal or Group identifier, … Terminal or Group identifier;*
TERMINAL_GROUP **G_*n*** = _ *Terminal or Group identifier, … Terminal or Group identifier;*

Multiple TERMINAL_GROUP definition syntax:

TERMINAL_GROUP {
G_*n* = _*Terminal or Group identifier, … Terminal or Group identifier;*
G_*n* = _ *Terminal or Group identifier, … Terminal or Group identifier;*
}

where:

8.4.6.2 G_*n*

Unique TERMINAL_GROUP identifier, where "***n***" is the actual TERMINAL_GROUP number (Integer, as 7.1.3.4). Note that the underscore character is optional, it is used for clarity only.

8.4.6.3 Terminal or Group identifier

Two or more terminal or terminal-group identifier names, as

- Any terminal identifier name must be pre-declared in a TERMINAL statement, refer to 8.4.4.1. A terminal identifier name may occur only once within a single group assignment.
- Any terminal-group identifier name must be pre-declared in a TERMINAL_GROUP statement, refer to 8.4.6.2. A terminal-group identifier name may occur only once within a single group assignment and assignment circularity or recursion is not permitted.

Example

```
TERMINAL_GROUP G_1 = T_10, T_11, T_12;
TERMINAL_GROUP G_2 = T_15, T_16, T_12;

TERMINAL_GROUP  {
     G_3 = T_1, T_2;
     G_4 = T_4, T_5;
     G_5 = G_1, T_3;
     G_6 = G_3, G_4, T_11;
}
```

Notes	In all cases, a single terminal identifier shall only occur with a group once. This is also relevant when considering the expansion of a terminal-group. Refer to Annex B.

8.4.7 PERMUTABLE Parameter

Parameter Name	PERMUTABLE
Parameter Type	Structure
Parameter Function	The PERMUTABLE structure assigns a list of terminals or terminal groups that are permutable. The list can comprise terminal identifiers (as 8.4.5.1) or terminal-group identifiers (as 8.4.6.2) but not a mixture of the two types of identifier. Each PERMUABLE assignment declares that the terminals or terminal groups listed are exchangeable and/or swappable in terms of functionality. This is to facilitate alternative connection and routing by CAD software. Refer PERMUTABLE Rules (see 8.4.7.1) for detail. Refer to Annex B for further explanation.
Parameter Values	The PERMUTABLE parameter statement shall consist of either a list of two or more terminal identifiers (see 8.4.6.1) or two or more terminal-group identifiers (see 8.4.6.1), but shall not contain a mixture of terminal and terminal-group identifiers.
Dependencies	TERMINAL (see 8.4.5) TERMINAL_GROUP (see 8.4.6)
Notes	In all cases, a single terminal identifier shall only occur with a group once. This is relevant when considering the expansion of a terminal-group. For correct permutation of terminal-groups within a single PERMUTABLE assignment, it is mandatory that the sequence of terminals within each group corresponds directly to the sequence of terminals of the other terminal-groups within the assignment.
Reference	Annex B.

8.4.7.1 PERMUTABLE Rules

8.4.7.1.1 Content

The elements forming a permutation may be terminal identifiers or group identifiers but not a mixture of the two. A permutation must contain at least two elements.

8.4.7.1.2 Uniqueness

All elements within a permutation must be unique. A permutation may not contain groups that refer directly or indirectly to the same element.
Each element within a permutation must contain the same number of Terminal identifiers (whether within Terminal Groups or not) as each other.

8.4.7.1.3 Ordering

The order of elements within a permutation is not important.

Single PERMUTABLE definition syntax:

PERMUTABLE **P_*n*** = *Terminal identifier, … Terminal identifier;*
PERMUTABLE **P_*n*** = *Terminal identifier, … Terminal identifier;*
PERMUTABLE **P_*n*** = *Group identifier, … Group identifier;*
PERMUTABLE **P_*n*** = *Group identifier, … Group identifier;*

Multiple PERMUTABLE definition syntax:

PERMUTABLE {
P_*n* = *Terminal identifier, … Terminal identifier;*
P_*n* = *Terminal identifier, … Terminal identifier;*
P_*n* = *Group identifier, … Group identifier;*
P_*n* = *Group identifier, … Group identifier;*
}

Example

```
PERMUTABLE P_1 = T_1, T_2, T_3;
PERMUTABLE P_2 = GP1, GP2, GP3, GP5, GP4;

PERMUTABLE {
     P_1 = T_1, T_2;
     P_2 = T_4, T_5;
     P_3 = T_9, T_10;
     P_4 = T_12, T_13;
     P_5 = G_5, G_6, G_7, G_8;
}
```

8.5 MATERIAL DATA

8.5.1 TERMINAL_MATERIAL Parameter

Parameter Name	**TERMINAL_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the material used for the final terminations on the device.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`TERMINAL_MATERIAL = "Al";` `TERMINAL_MATERIAL = "Copper";` `TERMINAL_MATERIAL = "SAC405";`

8.5.2 TERMINAL_MATERIAL_STRUCTURE Parameter

Parameter Name	TERMINAL_MATERIAL_STRUCTURE
Parameter Type	Variable
Parameter Function	Specifies the sequential structure of the materials used to build-up the terminations on the device.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`TERMINAL_MATERIAL_STRUCTURE = "Al-Ti-Ni-Au";`

8.5.3 DIE_SEMICONDUCTOR_MATERIAL Parameter

Parameter Name	**DIE_SEMICONDUCTOR_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the semiconductor material, which is used for fabrication of the die.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`DIE_SEMICONDUCTOR_MATERIAL = "Insulator";` `DIE_SEMICONDUCTOR_MATERIAL = "Silicon";` `DIE_SEMICONDUCTOR_MATERIAL = "Sapphire";`
Notes	This parameter is only relevant if the die bulk material is different from the die substrate material.

8.5.4 DIE_SUBSTRATE_MATERIAL Parameter

Parameter Name	**DIE_SUBSTRATE_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the bulk or substrate material on which the die is made where it differs from the DIE_SEMICONDUCTOR_MATERIAL.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DIE_SUBSTRATE_MATERIAL = "Silicon";`
	`DIE_SUBSTRATE_MATERIAL = "Insulator";`
	`DIE_SUBSTRATE_MATERIAL = "Sapphire";`

8.5.5 DIE_SUBSTRATE_CONNECTION Parameter

Parameter Name	**DIE_SUBSTRATE_CONNECTION**
Parameter Type	Variable
Parameter Function	Specifies the electrical connection for the substrate, if any. By fabrication, the substrate may be already electrically connected, which should also be stated. Where a substrate connection should be made, the potential point for connection shall also be stated.
Parameter Values	One or more textual string data, as 7.1.3.1 The first parameter shall conform to values given in Table 4
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DIE_SUBSTRATE_CONNECTION = "CONN","Ground";`
	`DIE_SUBSTRATE_CONNECTION = "N/A";`
	`DIE_SUBSTRATE_CONNECTION = "ISOL";`
	`DIE_SUBSTRATE_CONNECTION = "OPT","Most Negative";`
	`DIE_SUBSTRATE_CONNECTION = "OPT","Most Positive";`
	`DIE_SUBSTRATE_CONNECTION = "CONN","Digital Vdd";`
	`DIE_SUBSTRATE_CONNECTION = "OPT","Analog ground";`

Table 4 – Substrate Connection Parameters

Parameter Value	Function
CONN	**Must** be electrically connected
ISOL	**Must** be electrically isolated
OPT	May be optionally connected
N/A	Not Applicable
N/K	Not Known (unknown)

Notes When either "CONN" or "OPT" is given as the first parameter value, the second parameter value shall be comprehensively stated, sufficiently adequate to be understood for electrical connectivity. Recommended parameter values are: "Most Positive", "Most Negative", "GND", "AGND", "DGND", "VSS", "VDD", "VEE", "VCC", "AVSS", "AVDD", "AVEE", "AVCC", "DVSS", "DVDD", "DVEE", "DVCC","VBIAS", or "SPECIAL".
Alternatively, a specific TERMINAL name "**T_n**", as described in 8.4.5 may be used as the second parameter value for direct connection to the substrate.

8.5.6 DIE_PASSIVATION_MATERIAL Parameter

Parameter Name	**DIE_PASSIVATION_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the die surface passivation material.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DIE_PASSIVATION_MATERIAL = "Silicon Nitride";`
	`DIE_PASSIVATION_MATERIAL = "Oxy-Nitride";`
	`DIE_PASSIVATION_MATERIAL = "Polyimide";`

8.5.7 DIE_BACK_DETAIL Parameter

Parameter Name	**DIE_BACK_DETAIL**
Parameter Type	Variable
Parameter Function	Specifies the detail finish to the die backside, relevant for different attachment methods / package styles, to include both finish and material.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DIE_BACK_DETAIL = "Sawn";`
	`DIE_BACK_DETAIL = "Au-Metallized";`
	`DIE_BACK_DETAIL = "Backlapped";`
	`DIE_BACK_DETAIL = "Polished";`
	`DIE_BACK_DETAIL = "Gold Metal";`
	`DIE_BACK_DETAIL = "None";`

8.6 ELECTRICAL AND THERMAL RATING DATA

8.6.1 MAX_TEMP Parameter

Parameter Name	**MAX_TEMP**
Parameter Type	Variable
Parameter Function	Specifies the maximum temperature to which the device may be exposed during any part of die attach, device soldering or other manufacturing process.
Parameter Values	Real, as 7.1.3.3
Parameter Units	^{0}C, Celsius
Limitations	Shall be declared only once within a single DEVICE block.
Example	`MAX_TEMP = 280;`

8.6.2 MAX_TEMP_TIME Parameter

Parameter Name	**MAX_TEMP_TIME**
Parameter Type	Variable
Parameter Function	Specifies the maximum time for which the device may be exposed to the maximum temperature during any part of die attach, device soldering or other manufacturing process.
Parameter Values	Real, as 7.1.3.3
Parameter Units	Seconds
Dependencies	MAX_TEMP
Limitations	Shall be declared only once within a single DEVICE block.
Example	`MAX_TEMP_TIME = 10;`

8.6.3 POWER_RANGE Parameter

Parameter Name	**POWER_RANGE**
Parameter Type	Variable
Parameter Function	Specifies the power likely to be dissipated by the device.
Parameter Values	Real, as 7.1.3.3
Parameter Units	Watts
Limitations	Shall be declared only once within a single DEVICE block.
Example	`POWER_RANGE = 0.92;`
Notes	This parameter should be used with caution, as without fully specifying all measurement conditions, use of this value can be misunderstood. The value given should indicate the likely maximum power dissipated under "typical" worst-case conditions.

8.6.4 TEMPERATURE_RANGE Parameter

Parameter Name	**TEMPERATURE_RANGE**
Parameter Type	Variable
Parameter Function	Specifies the operation and specification range of the die.
Parameter Values	Minimum (Real, in ^{0}C), Maximum (Real, in ^{0}C), as clause 7.1.3.3

Parameter Units	^{0}C, Celsius
Limitations	Shall be declared only once within a single DEVICE block.
Example	`TEMPERATURE_RANGE = -40, 90;`
Notes	For use with bare die, this parameter should be used with caution, and is only intended to indicate the operational or specified temperature “grade” of the equivalent packaged part.

8.7 SIMULATION DATA

8.7.1 Simulator MODEL FILE Parameter

Parameter Name	**SIMULATOR_*simulator*_MODEL_FILE**
Parameter Type	Variable
Parameter Function	Specifies the name of the model file for use with *simulator*.
Parameter Values	Textual name data, as 7.1.3.2
Limitations	Shall be declared only once within a DEVICE block per *simulator* type.
Example	`SIMULATOR_SPICE_MODEL_FILE = “BC109.MOD”;` `SIMULATOR_SPECTRE_MODEL_FILE = “RTBAA1.S”;`
Notes	Only file names, without relative or absolute path names, shall be used.
Reference	Annex D.

8.7.2 Simulator MODEL FILE DATE Parameter

Parameter Name	**SIMULATOR_*simulator*_MODEL_FILE_DATE**
Parameter Type	Variable
Parameter Function	Specifies the creation or validation date of the model file.
Parameter Values	Date, as 7.1.3.5
Limitations	Shall be declared only once within a DEVICE block per *simulator* type.
Example	`SIMULATOR_SPICE_MODEL_FILE_DATE=”19951021”;` `SIMULATOR_VHDL_MODEL_FILE_DATE=”1993-05-17”;`
Notes	This date should match the date of the actual model file, to indicate that the correct model file is being used, and for use within a library make / build function.
Reference	Annex D.

8.7.3 Simulator NAME Parameter

Parameter Name	**SIMULATOR_*simulator*_NAME**
Parameter Type	Variable
Parameter Function	Specifies the name of either the generic or specific *simulator* to which the model file is appropriate.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a DEVICE block per *simulator* type.
Example	`SIMULATOR_SPICE_NAME   = “pSpice”;` `SIMULATOR_VERILOG_NAME  = “Verilog-XL”;`
Reference	Annex D.

8.7.4 Simulator VERSION Parameter

Parameter Name	**SIMULATOR_*simulator*_VERSION**
Parameter Type	Variable
Parameter Function	Specifies the version number of the *simulator* as specified by the SIMULATOR_*simulator*_NAME parameter, which was used to verify the model data.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a DEVICE block per *simulator* type.
Example	`SIMULATOR_SPICE_VERSION   = “4.0.1”;` `SIMULATOR_VERILOG_VERSION  = “1.7B”;`
Reference	Annex D.

8.7.5 Simulator COMPLIANCE Parameter

Parameter Name	**SIMULATOR_*simulator*_COMPLIANCE**
Parameter Type	Variable
Parameter Function	Specifies the minimum compliance level of the *simulator* required to both accurately reproduce and correlate with simulation results.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a DEVICE block per *simulator* type.

Example	`SIMULATOR_VHDL_COMPLIANCE  = "VHDL '93";`
	`SIMULATOR_SPICE_COMPLIANCE  = "2G6";`

Reference	Annex D.

8.7.6 Simulator TERM_GROUP Parameter

Parameter Name	**SIMULATOR_*simulator*_TERM_GROUP**
Parameter Type	Variable
Parameter Function	Specifies the group or terminals to which the simulator model applies. If missing, the assumption is that the simulator model applies to the whole device.
Parameter Values	A list of terminals (as 8.4.5.1) and/or terminal groups (as 8.4.6.2)
Limitations	The terminals or terminal groups referenced shall have already been declared in either a TERMINAL or TERMINAL_GROUP parameter. Shall be declared only once within a DEVICE block per *simulator* type.

Example	`SIMULATOR_SPICE_TERM_GROUP = T_1, T_2, G_1;`
	`SIMULATOR_IBIS_TERM_GROUP = G_5, G_8, T_11, T_33;`

Reference	Annexes B and D.

8.8 HANDLING, PACKING, STORAGE and ASSEMBLY DATA

8.8.1 DELIVERY_FORM Parameter

Parameter Name	**DELIVERY_FORM**
Parameter Type	Variable
Parameter Function	Specifies the form in which the die is delivered.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DELIVERY_FORM = "Wafer";`
	`DELIVERY_FORM = "Die";`
	`DELIVERY_FORM = "Bare Die in Waffle Tray";`
	`DELIVERY_FORM = "Sawn Wafer";`
	`DELIVERY_FORM = "Tray";`
	`DELIVERY_FORM = "Bandoleer", "Waffle";`

8.8.2 PACKING_CODE Parameter

Parameter Name	**PACKING_CODE**
Parameter Type	Variable
Parameter Function	Specifies the form of primary packing used for the supply of devices.
Parameter Values	Textual string data, as 7.1.3.1

Example	`PACKING_CODE = "WAFFLE";`
	`PACKING_CODE = "GEL_PACK";`
	`PACKING_CODE = "Bandoleer";`

8.8.3 ASSEMBLY Parameters

Parameter Name	ASSY_ *text-identifier*
Parameter Type	Variable
Parameter Function	General textual parameter, to enable the user to include relevant and identified assembly techniques, parameters and operations.
Parameter Values	Textual string data, as 7.1.3.1
Examples	`ASSY_PROCESS_LIMITATIONS = "NONE";`
Notes	Recognised *ASSY_identifiers* are:

```
8.8.3.1    ASSY_PROCESS_LIMITATIONS
8.8.3.2    ASSY_STORAGE_LIMITATIONS
8.8.3.3    ASSY_ASSEMBLY_LIMITATIONS
8.8.3.4    ASSY_TEMPERATURE_LIMITATIONS
8.8.3.5    ASSY_BONDING_METHODS
8.8.3.6    ASSY_BONDING_MATERIALS
8.8.3.7    ASSY_ATTACH_METHODS
8.8.3.8    ASSY_ATTACH_MATERIALS
8.8.3.9    ASSY_GENERAL_REQUIREMENTS
8.8.3.10   ASSY_HANDLING_REQUIREMENTS
8.8.3.11   ASSY_PACKING_REQUIREMENTS
8.8.3.12   ASSY_STORAGE_REQUIREMENTS
8.8.3.13   ASSY_SHIPPING_REQUIREMENTS
```

8.9 WAFER SPECIFIC DATA

8.9.1 WAFER_SIZE Parameter

Parameter Name	**WAFER_SIZE**
Parameter Type	Variable
Parameter Function	Specifies the wafer diameter.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`WAFER_SIZE = “6 inch”;`
	`WAFER_SIZE = “150mm”;`

Notes	As this is a dimensional value, but related to the fabrication and not the die size, the WAFER_SIZE parameter may be in units different to those defined by the GEOMETRIC_UNITS parameter, hence the data is presented as free-form text.

8.9.2 WAFER_THICKNESS Parameter

Parameter Name	**THICKNESS**
Parameter Type	Variable
Parameter Function	Determines the thickness (Z-dimension) of the wafer.
Parameter Values	Real Z-dimension value, as in 7.1.3.3
Dependencies	GEOMETRIC_UNITS
Limitations	Shall be declared only once within a single DEVICE block.

Example	`WAFER_THICKNESS = 10.5;`
	`WAFER_THICKNESS = 80;`

Notes	Dimension values are in GEOMETRIC_UNITS.

8.9.3 WAFER_THICKNESS_TOLERANCE Parameter

Parameter Name	**WAFER_THICKNESS_TOLERANCE**
Parameter Type	Variable
Parameter Function	Specifies the tolerance(s) of the WAFER_THICKNESS parameter that may be expected due to normal process variations.
Parameter Values	Real in GEOMETRIC_UNITS, as 7.1.3.3
Dependencies	GEOMETRIC_UNITS, WAFER_THICKNESS
Limitations	Shall be declared only once within a single DEVICE block.

Example	`WAFER_THICKNESS_TOLERANCE = 2.5;`
	`WAFER_THICKNESS_TOLERANCE = -1.2,1.5;`

Notes	One or two values may be given:- Where a single tolerance value is given, the tolerance shall be taken as an unsigned ± value. Where two tolerance values are given:- the first value shall be taken as the minimum, and the second value shall be taken as the maximum.

8.9.4 WAFER_DIE_STEP_SIZE Parameter

Parameter Name	**WAFER_DIE_STEP_SIZE**
Parameter Type	Variable
Parameter Function	Determines the X- and Y- step size dimensions, in GEOMETRIC_UNITS, for the die on wafer, as required by mechanical handling equipment.
Parameter Values	Real X-dimension, Real Y-dimension, (as 7.1.3.3)
Dependencies	GEOMETRIC_UNITS, GEOMETRIC_VIEW
Limitations	Shall be declared only once within a single DEVICE block.

Example	`WAFER_DIE_STEP_SIZE = 280, 530;`
	`WAFER_DIE_STEP_SIZE = 1500, 1500;`

Reference	Annex H.

8.9.5 WAFER_GROSS_DIE_COUNT Parameter

Parameter Name	**WAFER_GROSS_DIE_COUNT**
Parameter Type	Variable
Parameter Function	Specifies the number of whole and viable gross die (of the die type in question) available on the wafer.
Parameter Values	Integer, as 7.1.3.4
Limitations	Shall be declared only once within a single DEVICE block.
Example	`WAFER_GROSS_DIE_COUNT = 2027;`
Notes	This parameter value is only intended to give an indication of the number of relevant and viable die per wafer, and shall have no other implication.
Reference	Annex H.

8.9.6 WAFER_INDEX Parameter

Parameter Name	**WAFER_INDEX**
Parameter Type	Variable
Parameter Function	Specifies the type of index feature on a wafer which acts as a reference feature and its orientation with respect to the X-axis for the die. The first parameter determines the index feature type, and the second parameter specifies the approximate angular relationship between the assumed X-axis for the die and the index feature on the wafer. The angle shall be given, in degrees counted clockwise, of the index feature, taking the X-axis of the die as the reference.
Parameter Values	Textual string data, as 7.1.3.1, with value “Flat” or “Notch”, followed by a single integer value, in units of degrees, from 0 to 359, as 7.1.3.4.
Limitations	Shall be declared only once within a single DEVICE block.

Example	`WAFER_INDEX = “Flat”,90;`
	`WAFER_INDEX = “Notch”,0;`

Notes	This parameter is only intended as a guide, and so integer approximations shall be acceptable. Negative angle values are not permitted. Where more than one flat exists on the wafer, the orientation is with respect to the major, or prime, flat. This may also indicate the crystal [110] direction.
Reference	Annex H.

8.9.7 WAFER_RETICULE_STEP_SIZE Parameter

Parameter Name	**WAFER_RETICULE_STEP_SIZE**
Parameter Type	Variable
Parameter Function	Specifies X- and Y- step and repeat dimensions for a single reticule.
Parameter Values	Real X-dimension, Real Y-dimension, (as 7.1.3.3)
Dependencies	GEOMETRIC_UNITS, GEOMETRIC_VIEW
Limitations	Shall be declared only once within a single DEVICE block.

Example	`WAFER_RETICULE_STEP_SIZE = 2500, 8000;`
	`WAFER_RETICULE_STEP_SIZE = 15000, 27500;`

Notes	This parameter is relevant mainly for MPW (Multi Project Wafers), or instances where the WAFER_DIE_STEP_SIZE becomes invalid due to a non-integer relationship between reticule size and die size.
Reference	Annex H.

8.9.8 WAFER_RETICULE_GROSS_DIE_COUNT Parameter

Parameter Name	**WAFER_RETICULE_GROSS_DIE_COUNT**
Parameter Type	Variable
Parameter Function	Specifies the number of whole and viable gross die (of the die type in question) emplaced within a single reticule.
Parameter Values	Integer, as 7.1.3.4
Limitations	Shall be declared only once within a single DEVICE block.
Example	`WAFER_RETICULE_GROSS_DIE_COUNT = 40;`
Notes	This parameter is relevant mainly for MPW (Multi Project Wafers), or instances where there is more than one die type in the reticule. This parameter value is only intended to give an indication of the number of relevant and viable die per reticule, and shall have no other implication.
Reference	Annex H.

8.9.9 WAFER_INK Parameters

Parameter Name	WAFER_INK_*text-identifier*
Parameter Type	Variable
Parameter Function	General textual parameter, to enable the user to include relevant parameters relating to the wafer dot/ink size and colour. This is of particular relevance for visual systems for die selection, sorting and inspection.
Parameter Values	Textual string data, as 7.1.3.1

Examples

```
WAFER_INK_COLOUR = "BLACK";
WAFER_INK_LOCATION = "CENTRAL";
WAFER_INK_SIZE_MAX = "0.8mm";
WAFER_INK_SORT_COLOUR = "BIN1, RED, UPPER RIGHT";
WAFER_INK_SORT_COLOUR = "BIN2, GREEN, LOWER LEFT";
```

Notes: Recognised *WAFER_INK_identifiers* are:

```
8.9.9.1    WAFER_INK_COLOUR
8.9.9.2    WAFER_INK_SIZE
8.9.9.3    WAFER_INK_SIZE_TOL
8.9.9.4    WAFER_INK_SIZE_MAX
8.9.9.5    WAFER_INK_LOCATION
8.9.9.6    WAFER_INK_LOCATION_TOL
8.9.9.7    WAFER_INK_HEIGHT_MAX
8.9.9.8    WAFER_INK_SORT_COLOUR
```

8.10 BUMP TERMINATION SPECIFIC DATA

Any data given that is pertinent to a "bumped" device (a device using added bump connection structures) can only reflect the bump parameters at the time of bump formation, and not necessarily the final parameters. Due to the nature of final bump connection, the bump parameters will be altered and deformed during the attachment process. Therefore all geometric and metallurgical parameters given should be considered for "virgin" bumps.

8.10.1 BUMP_MATERIAL Parameter

Parameter Name	**BUMP_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the metallurgical material of the bump contact.
Parameter Values	Textual string data, as clause 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`BUMP_MATERIAL = "Copper";`
	`BUMP_MATERIAL = "Molybdenum Telluride";`
	`BUMP_MATERIAL = "Au";`
	`BUMP_MATERIAL = "SAC415";`
	`BUMP_MATERIAL = "Pb-Sn";`

Notes	Required only for device types with bump connection structures. Where a device has both bumped and non-bumped terminals, TERMINAL_MATERIAL should be used to describe the non-bumped terminals, and BUMP_MATERIAL should be used to describe the bumped terminal.

8.10.2 BUMP_HEIGHT Parameter

Parameter Name	**BUMP_HEIGHT**
Parameter Type	Variable
Parameter Function	Specifies the bump contact height above the passivation surface.
Parameter Values	Numeric real, in GEOMETRIC_UNITS, as 7.1.3.3
Dependencies	GEOMETRIC_UNITS
Limitations	Shall be declared only once within a single DEVICE block.
Example	`BUMP_HEIGHT = 150.0;`
Notes	Required only for device types with bump connection structures. The height of the bump can only be stated before the bump is modified by attachment and/or permanent connection.

8.10.3 BUMP_HEIGHT_TOLERANCE Parameter

Parameter Name	**BUMP_HEIGHT_TOLERANCE**
Parameter Type	Variable
Parameter Function	Specifies the tolerance of bump contact height above the passivation surface.
Parameter Values	Numeric real, in GEOMETRIC_UNITS, as 7.1.3.3
Dependencies	GEOMETRIC_UNITS, BUMP_HEIGHT
Limitations	Shall be declared only once within a single DEVICE block.

Example	`BUMP_HEIGHT_TOLERANCE = 35.0;`
	`BUMP_HEIGHT_TOLERANCE = -10.0,25.0;`

Notes	One or two values may be given:- Where a single tolerance value is given, the tolerance shall be taken as an unsigned ± value. Where two tolerance values are given:- the first value shall be taken as the minimum, and the second value shall be taken as the maximum. Required only for device types with bump connection structures.

8.10.4 BUMP_SHAPE Parameter

Parameter Name	**BUMP_SHAPE**
Parameter Type	Variable
Parameter Function	Specifies the shape of bump the contact height.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`BUMP_SHAPE="Pyramidical";`
	`BUMP_SHAPE="Ball";`

Notes	Required only for device types with bump connection structures. The shape of the bump can only be stated before the bump is modified by attachment and/or permanent connection.

8.10.5 BUMP_SIZE Parameter

Parameter Name	**BUMP_SIZE**
Parameter Type	Variable
Parameter Function	Specifies the size of bump X-Y co-ordinates.
Parameter Values	Numeric real, in GEOMETRIC_UNITS, as 7.1.3.3

Dependencies	GEOMETRIC_UNITS
Limitations	Shall be declared only once within a single DEVICE block.
Example	`BUMP_SIZE= "150,150";`
Notes	Required only for device types with bump connection structures. The size of the bump can only be stated before the bump is modified by attachment and/or permanent connection.

8.10.6 BUMP_SPECIFICATION_DRAWING Parameter

Parameter Name	**BUMP_SPECIFICATION_DRAWING**
Parameter Type	Variable
Parameter Function	Specifies the name(s) of the bump specification drawing file(s).
Parameter Values	Textual name data, as 7.1.3.2
Example	`BUMP_SPECIFICATION_DRAWING = "BumpShape1.JPG";` `BUMP_SPECIFICATION_DRAWING = "BS1.JPG", "BS2.TIF";`
Notes	Only file names, without relative or absolute path names, shall be used. Required only for device types with bump connection structures. Any drawing of the bump can only reflect the state of the bump before being modified by attachment and/or permanent connection. Multiple parameters (specification drawing files) may be introduced within a single declaration, or multiple declarations maybe be employed to the same effect. There is no scaling or positional data requirement.
Reference	Annex I, Note 3 and 4.

8.10.7 BUMP_ATTACHMENT_METHOD Parameter

Parameter Name	**BUMP_ATTACHMENT_METHOD**
Parameter Type	Variable
Parameter Function	Specifies the type of attachment method required or suggested for this type of bump.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`BUMP_ATTACHMENT_METHOD = "Thermocompression";` `BUMP_ATTACHMENT_METHOD = "Solder Reflow";`
Notes	Required only for device types with bump connection structures.

8.11 MINIMALLY PACKAGED DEVICE (MPD) SPECIFIC DATA

8.11.1 MPD_PACKAGE_MATERIAL Parameter

Parameter Name	**MPD_PACKAGE_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the package body material used as final encapsulation of the MPD.
Parameter Values	Textual string data as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`MPD_PACKAGE_MATERIAL = "Plastic";` `MPD_PACKAGE_MATERIAL = "Epoxy";` `MPD_PACKAGE_MATERIAL = "Ceramic";` `MPD_PACKAGE_MATERIAL = "Metal Can";`
Notes	Generally required for assembly purposes.

8.11.2 MPD_PACKAGE_STYLE Parameter

Parameter Name	**MPD_PACKAGE_STYLE**
Parameter Type	Variable
Parameter Function	Specifies the code for the package style as defined in a standard or as given by the manufacturer.
Parameter Values	Textual string data as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`MPD_PACKAGE_STYLE = "SOT-23";`
	`MPD_PACKAGE_STYLE = "MO";`
	`MPD_PACKAGE_STYLE = "TSOP-48";`

8.11.3 MPD_CONNECTION_TYPE Parameter

Parameter Name	**MPD_CONNECTION_TYPE**
Parameter Type	Variable
Parameter Function	Specifies the connection method to the MPD, such as lead, bump etc.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`MPD_CONNECTION_TYPE = "Bump";`
	`MPD_CONNECTION_TYPE = "TAB Lead";`

8.11.4 MPD_MSL_LEVEL Parameter

Parameter Name	**MPD_MSL_LEVEL**
Parameter Type	Variable
Parameter Function	Specifies the Moisture Sensitivity Level of the MPD package, in accordance with IPC/JEDEC J-STD-033B:2007 – Handling, Packing, Shipping and Use of Moisture/Reflow Sensitive Surface Mount Devices, 2005.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`MPD_MSL_LEVEL = "3";`

8.11.5 MPD_PACKAGE_DRAWING Parameter

Parameter Name	**MPD_PACKAGE_DRAWING**
Parameter Type	Variable
Parameter Function	Specifies the name(s) of the MPD package specification drawing file(s).
Parameter Values	Textual name data, as 7.1.3.2

Example	`MPD_PACKAGE_DRAWING = "PackageOutline.PDF";`
	`MPD_PACKAGE_DRAWING = "PO1.JPG","PO1A.GIF";`

Notes	Only file names, without relative or absolute path names, shall be used. Multiple parameters (package drawing files) may be introduced within a single declaration, or multiple declarations maybe be employed to the same effect. There is no scaling or positional data requirement.
Reference	Annex I, Notes 3 and 4.

8.12 QUALITY, RELIABILITY and TEST DATA

8.12.1 QUALITY Parameters

Parameter Name	**QUAL_ *identifier***
Parameter Type	Variable
Parameter Function	General text parameter, to enable the user to include relevant and identified quality and reliability parameters.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Each identifier shall be declared only once within a single DEVICE block.
Examples	`QUAL_OUTGOING_QUALITY_LEVEL = "….";`
Notes	Recognised *QUAL_identifiers* are:

```
8.12.1.1   QUAL_OUTGOING_QUALITY_LEVEL
8.12.1.2   QUAL_OUTGOING_QUALITY_UNITS
8.12.1.3   QUAL_OUTGOING_QUALITY_DESCRIPTION
8.12.1.4   QUAL_RELIABILITY_VALUE
8.12.1.5   QUAL_RELIABILITY_UNITS
8.12.1.6   QUAL_RELIABILITY_REFERENCE
```

```
8.12.1.7   QUAL_RELIABILITY_CONDITIONS
8.12.1.8   QUAL_RELIABILITY_CALC_METHOD
8.12.1.9   QUAL_STANDARDS_COMPLIANCE
```

8.12.2 TEST Parameters

Parameter Name	**TEST_ *identifier***
Parameter Type	Variable
Parameter Function	General text parameter, to enable the user to include relevant and identified test related parameters.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Each identifier shall be declared only once within a single DEVICE block.
Examples	`TEST_ADDITIONAL_REGUIREMENTS = “NONE”;` `TEST_SCREEN_COMPLIANCE = ”MIL-883B”;`
Notes	Recognised *TEST_identifiers* are:

```
8.12.2.1   TEST_ELECTRICAL_CONDITIONS
8.12.2.2   TEST_ADDITIONAL_SCREENING
8.12.2.3   TEST_TESTABILITY_FEATURES
8.12.2.4   TEST_ADDITIONAL_REQUIREMENTS
8.12.2.5   TEST_YIELD_CODE
8.12.2.6   TEST_FLOW
8.12.2.7   TEST_TEMP
8.12.2.8   TEST_SCREEN
8.12.2.9   TEST_SCREEN_COMPLIANCE
```

8.13 OTHER DATA

8.13.1 TEXT Parameters

Parameter Name	**TEXT_*-identifier***
Parameter Type	Variable
Parameter Function	General text parameter, to enable the user to include relevant and identified text not otherwise covered.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Each identifier shall be declared only once within a single DEVICE block.
Examples	`TEXT_SPECIAL_REQUIREMENTS = “N/A”;`
Notes	Recognised *TEXT_identifiers* are:

```
8.13.1.1   TEXT_PRODUCT_STATUS
8.13.1.2   TEXT_FORM_OF_SUPPLY
8.13.1.3   TEXT_SPECIAL_REQUIREMENTS
8.13.1.4   TEXT_SPECIFIC_REQUIREMENTS
8.13.1.5   TEXT_STORAGE_CONDITIONS
8.13.1.6   TEXT_STORAGE_DURATION
8.13.1.7   TEXT_LONGTERM_STORAGE
8.13.1.8   TEXT_ORIGINAL_MANUFACTURER
8.13.1.9   TEXT_ORIGINAL_DESIGN_DATE
```

8.14 CONTROL DATA

8.14.1 PARSE Parameters

Parameter Name	**PARSE_*-identifier***
Parameter Type	Variable
Parameter Function	Specific parameters for use as control flags or parameters by the parsing software only, these parameters have no DEVICE related meaning nor function.

Parameter Values	Textual string data, as 7.1.3.1
Notes	The PARSE_xxx parameters are only applicable to data occurring subsequent to their invocation; this allows the PARSE_xxx parameters to be changed during parsing throughout a single DEVICE block. It is therefore strongly advised that if the PARSE_xxx parameters are to be used, the initial invocation occurs before any other data within the device BLOCK. Multi-pass parsers must accommodate this feature. These parse parameters may be modified or overridden by other options within the parsing software, such as command-line options. Refer to Annex K for further details.

Examples

```
PARSE_MODE = STRICT;
PARSE_ERROR_REPORT = TERSE;
PARSE_ERROR_TRAP = ALL;
PARSE_IGNORE = NONE;
PARSE_DEFINE_PARAMETER = "A_New_Parameter";
PARSE_DEFINE_STRUCTURE = "A_New_Structure";
```

8.14.1.1 PARSE_MODE

The PARSE_MODE parameter selects the severity of rule and syntax checking within a device BLOCK. Refer to Annex K for more detail.

The recognised PARSE_MODE parameters are:

```
STRICT
RELAXED
ENHANCED
USER
```

8.14.1.2 PARSE_ERROR_REPORT

The PARSE_ERROR_REPORT parameter determines how the warnings and errors uncovered during parsing of the device BLOCK are reported. A cumulative report of warnings and errors is expected once all parsing has been completed, this feature is not affected by this parameter. Refer to Annex K for further details.

The recognised PARSE_ ERROR_REPORT parameters are:

OFF	No errors are reported during parsing
TERSE	Only errors are reported during parsing
VERBOSE	All warnings and errors are reported during parsing

8.14.1.3 PARSE_ERROR_TRAP

The PARSE_ERROR_TRAP parameter determines how the parsing software should proceed once an error has been uncovered. Refer to Annex K for further details.

The recognised PARSE_ ERROR_TRAP parameters are:

ALL	Parser should not halt on error.
FIRST	Parser should halt on first error.

8.14.1.4 PARSE_IGNORE

The PARSE_IGNORE parameter determines whether the data is checked or not. Extreme caution must be exercised when using this parameter. Refer to Annex K for further details.

The recognised PARSE_ IGNORE parameters are:

`NONE`	All parsing checks are performed.
`OFF`	All parsing checks are performed (as `NONE`)
`ALL`	All parsing checks are ignored.
`SYNTAX_ONLY`	Only syntactical errors and line-termination checks.

8.14.1.5 PARSE_DEFINE_PARAMETER

The PARSE_DEFINE_PARAMETER parameter allows for the introduction and definition of new data parameters by name prior to them being incorporated within a new release of the DDX syntax. Refer to Annex K for further details.

Example:

```
PARSE_DEFINE_PARAMETER = "My_New_Parameter";
PARSE_DEFINE_PARAMETER = "My_Second_Parameter";

My_New_Parameter = "some data";
My_Second_Parameter = "some other data";
```

If the “new” parameter name conflicts with parameter names or reserved words already in existence, a warning should be flagged up, and the “new” parameter treated as defined in the current DDX standard, i.e. the PARSE_DEFINE_PARAMETER should be flagged as a warning, then ignored. This will allow the parsing software to accept these “new” parameters once they have been ratified without re-writing the DDX file.

The “new” name shall follow the general guidelines as set out in Clauses 5 to 7, and will only be valid within that device BLOCK. The “new” parameter will then be checked for syntax and parsed as if it were a standard parameter.

The PARSE_DEFINE_PARAMETER may occur as many times as needed, each statement may only introduce a single parameter, and must precede the use of that parameter.

Until ratified and incorporated with a new release of the DDX standard, all values pertaining to the “new” parameter will be treated as textual string data, as 7.1.3.1.

8.14.1.6 PARSE_DEFINE_STRUCTURE

The PARSE_DEFINE_STRUCTURE parameter allows for the introduction of new data structures by name prior to them being incorporated within a new release of the DDX syntax. Refer to Annex K for further details.

Example:

```
PARSE_DEFINE_STRUCTURE = "A_New_Structure";
PARSE_DEFINE_STRUCTURE = "Another_Structure";

A_New_Structure {
  NEW_STRUCTURE = "some data", "some more data";
  NEW_STRUCTURE = "some data", "some more data";
}
```

```
Another_Structure {
  ANOTHER_NEW_STRUCT = “some data”, “some more data”;
  ANOTHER_NEW_STRUCT = “some data”, “some more data”;
}
```

If the “new” structure name conflicts with structure names or reserved words already in existence, a warning should be flagged up, and the “new” structure treated as defined in the current DDX standard, i.e. the PARSE_DEFINE_STRUCTURE should be flagged as a warning, then ignored. This will allow the parsing software to accept these “new” structures once they have been ratified without re-writing the DDX file.

The “new” name shall follow the general guidelines as set out in Clauses 5 to 7, and will only be valid within that device BLOCK. The “new” structure will then be checked for syntax and parsed as if it were a structure.

The PARSE_DEFINE_STRUCTURE may occur as many times as needed, each statement may only introduce a single structure, and must precede the use of that structure.

The “*Structure_indent_name*” may be referenced as any other structure name, but again not within existing defined structure parameters until ratified.

Until ratified and incorporated with a new release of the DDX standard, all individual values contained within the “new” structure will be treated as textual string data, as 7.1.3.1.

Annex A
(informative)

An example of a DDX DEVICE block

```
DEVICE 7995 bare_die {
#
# Initial header data, with block and device history.
#
BLOCK_CREATION_DATE = “2000-12-25”;
BLOCK_VERSION = 1.0;
MANUFACTURER = “Fuzziwuzz Logic Ltd.”;
FUNCTION = “Special gate”;
DATA_SOURCE = “GOOD-DIE database”;
DATA_VERSION = “Initial Issue A”;
VERSION = “1.2.2”;

#
# Declaration of geometric view, units, size etc.,
#
GEOMETRIC_UNITS = millimetre;
GEOMETRIC_VIEW = “top”;
SIZE = 1.312, 1.050;
SIZE_TOLERANCE = 0.00, 0.0005, 0.00 0.0005;
THICKNESS = 0.360;
THICKNESS_TOLERANCE = 0.00, 0.0007;
GEOMETRIC_ORIGIN = 0,0;

#
# Additional details of Die type and usage.
#
DI*E_NAME = “XXZ322”;
DIE_MASK_REVISION = “Mask 1.0”;
MAX_TEMP = 280;
POWER_RANGE = 0.500;
DIE_SUBSTRATE_MATERIAL = “Silicon”;
DIE_TERMINAL_MATERIAL = “Al”;
IC_TECHNOLOGY = “bipolar”;
DIE_SUBSTRATE_CONNECTION = “Ground”;

#
# Delivery details.
#
DIE_BACK_DETAIL  = “Back-Lapped”;
DIE_DELIVERY_FORM = “Die, Wafer”;
WAFER_SIZE = “4 inch”;

#
# Definition of the number of bond pad types, bond pads
# and connections.
#
TERMINAL_TYPE_COUNT    = 5;
TERMINAL_COUNT         = 8;
CONNECTION_COUNT       = 14;
#
# Definition of the bond pad shapes and dimensions.
#
TERMINAL_TYPE  {
      PADR1 = Rectangle, 0.144, 0.104;
      PADR2 = Rectangle, 0.264, 0.104;
      PADR3 = Rectangle, 0.084, 0.084;
      PADC1 = Circle, 0.100;
      PADP1 = Polygon, (-0.0175,-0.042),(-0.042,-0.0175),
      (-0.042, 0.0175),(-0.0175, 0.042),
      ( 0.0175, 0.042),( 0.042, 0.0175),
      ( 0.042,-0.0175),( 0.0175,-0.042);
      }

#
# Bond pad placement, naming, orientation and connectivity
# details.
#
TERMINAL {
      T1 = 1 ,PADC1,-0.550, 0.416,  0,VCCA   ,P;
      T2 = 3 ,PADP1,-0.502, 0.190,  0,INPUTA ,I;
      T3 = 4 ,PADP1,-0.502,-0.192,  0,INPUTB ,I;
      T4 = 7 ,PADC1,-0.399,-0.442,  0,GNDA   ,G;
      T5 = 8 ,PADR2, 0.498,-0.442,  0,GNDB   ,G;
      T6 = 11,PADR3, 0.511,-0.171,  0,OUTPUTA,O;
      T7 = 12,PADR3, 0.511, 0.171,  0,OUTPUTB,O;
      T8 = 14,PADR1, 0.558, 0.416,  0,VCCB   ,P;
      }

#
# Details of a supplied pSpice simulation model.
#
SIMULATOR_SPICE_MODEL_FILE = "SP7995.MOD";
SIMULATOR_SPICE_MODEL_FILE_DATE = "1997-09-17";
SIMULATOR_SPICE_NAME = "pSpice";
SIMULATOR_SPICE_VERSION = "4.0.1";
SIMULATOR_SPICE_COMPLIANCE = "2G6";

#
# Details of a supplied Spectre simulation model.
#
SIMULATOR_SPECTRE_MODEL_FILE = "SP7995.S";
SIMULATOR_SPECTRE_MODEL_FILE_DATE = "1998-11-05";
SIMULATOR_SPECTRE_NAME = "Spectre";
SIMULATOR_SPECTRE_VERSION = "4.2.1, 1992";
SIMULATOR_SPECTRE_COMPLIANCE = "2G6, Level-3";

#
# Details of reference fiducial, supplied as a JIF graphic file
#
FIDUCIAL_TYPE fiduc1 = "7995FID1.JIF", 0.072, 0.055;
FIDUCIAL F1 = fiduc1, -0.612, 0.470, 0;

# End of block
}
```

Annex B
(informative)

Groups and Permutation

A. Specific Rules for the TERMINAL_GROUP parameter

Rule A1. ***Content*** (refer to 8.4.6.1.1) – The elements forming a group may be terminal identifiers or group identifiers or any mixture of the two. A group must contain at least two elements.

Rule A2. ***Uniqueness*** (refer to 8.4.6.1.2) – All elements within a group must be unique. A group may not contain groups that refer directly or indirectly to the same element.

Rule A3. ***Ordering*** (refer to 8.4.6.1.3) – the order of elements within groups may be important. Where two or more groups are to be related in a permutation, then the elements within those groups must correspond across the groups.

Rule A4. ***Recursion*** (refer to 8.4.6.1.4) – A group may not contain itself nor may it contain any group that refers directly or indirectly to itself.

B. Specific Rules for the PERMUTABLE parameter

Rule B1. ***Content*** (refer to 8.4.7.1.1) – The elements forming a permutation may be terminal identifiers or group identifiers but not a mixture of the two. A permutation must contain at least two elements.

Rule B2. ***Uniqueness*** (refer to 8.4.7.1.2) – All elements within a permutation must be unique. A permutation may not contain groups that refer directly or indirectly to the same element. Each element within a permutation must contain the same number of Terminal identifiers (whether within Terminal Groups or not) as each other.

Rule B3. ***Ordering*** (refer to 8.4.7.1.3) – the order of elements within a permutation is not important.

The simplest method of describing the use and function of the TERMINAL_GROUP and PERMUTABLE parameters is by analysing a simple example. Here, a fragment of a DDX file for a 7400 device is given. The 7400 device is a quad 2-input NAND gate, and so for functionality, all four NAND gates are exchangeable. Throughout the analysis we refer to terminals, but for a packaged part the user may consider the term “pins” to be more appropriate.

```
DEVICE 74ACT00 bare_die {

  BLOCK_CREATION_DATE = "13/02/2006";
  BLOCK_VERSION = 1.0;
  DEVICE_NAME = "74ACT00";
  MANUFACTURER = "Fuzziwuzz Logic Ltd";
  FUNCTION = "Quad two-input advanced CMOS NAND gate";
  DEVICE_FORM = "bare_die";
  DATA_SOURCE = "GOOD-DIE Project database";
  VERSION = "1.3.0";
  GEOMETRIC_UNITS = micrometre;
  GEOMETRIC_VIEW = "top";
  SIZE = 1067, 1143;
  THICKNESS = 356;
  GEOMETRIC_ORIGIN = 0,0;
  DIE_NAME = "74ACT00";
  DIE_MASK_REVISION = "Mask T";
  MAX_TEMP = 150;
  POWER_RANGE = 0.2;
  IC_TECHNOLOGY = "CMOS";
  DIE_SEMICONDUCTOR_MATERIAL = "silicon";
  DIE_SUBSTRATE_CONNECTION = "CONN, Vcc";
  DIE_DELIVERY_FORM = "Die, wafer";
  TERMINAL_TYPE_COUNT = 1;

  TERMINAL_TYPE  {
      PADR1 = Rectangle, 97, 97;
      }

  TERMINAL_COUNT = 14;

  TERMINAL  {
      T_1 = 1, PADR1, -385, 422, 0, A1, I;
      T_2 = 2, PADR1, -385, 176, 0, B1, I;
      T_3 = 3, PADR1, -385, 11, 0, Y1, O;
      T_4 = 4, PADR1, -385, -236, 0, A2, I;
      T_5 = 5, PADR1, -208, -423, 0, B2, I;
      T_6 = 6, PADR1, -43, -423, 0, Y2, O;
      T_7 = 7, PADR1, 123, -423, 0, GND, G;
      T_8 = 8, PADR1, 385, -423, 0, Y3, O;
      T_9 = 9, PADR1, 385, -166, 0, B3, I;
      T_10 = 10, PADR1, 385, -1, 0, A3, I;
      T_11 = 11, PADR1, 385, 164, 0, Y4, O;
      T_12 = 12, PADR1, 385, 423, 0, B4, I;
      T_13 = 13, PADR1, 38, 423, 0, A4, I;
      T_14 = 14, PADR1, -129, 423, 0, VCC, P;
      }

TERMINAL_GROUP  {
      NAND_INA = T_1, T_2;
      NAND_INB = T_4, T_5;
      NAND_INC = T_9, T_10;
      NAND_IND = T_12, T_13;
      NAND_A = NAND_INA, T_3;
      NAND_B = NAND_INB, T_6;
      NAND_C = NAND_INC, T_8;
      NAND_D = NAND_IND, T_11;
      }

PERMUTABLE {
      P_1 = T_1, T_2;
      P_2 = T_4, T_5;
      P_3 = T_9, T_10;
      P_4 = T_12, T_13;
      P_5 = NAND_A, NAND_B, NAND_C, NAND_D;
      }
}
```

The 7400 device has 4 identical gates, Gates A to D, with the following characteristics:

7400 Device	INPUT1	INPUT2	OUTPUT
Gate A	T_1	T_2	T_3
Gate B	T_4	T_5	T_6
Gate C	T_9	T_10	T_8
Gate D	T_12	T_13	T_11

Regarding the DDX fragment above, we see that `TERMINAL_GROUP NAND_INA` groups the inputs for Gate A together, `NAND_INB` for Gate B and so on ...

`TERMINAL_GROUP NAND_A` similarly groups the inputs for Gate A along with the relevant output terminal, and again `NAND_B` to `NAND_D` follow the Gates B to D in similar fashion. Note that the mixing of terminal-groups and terminals within the same statement (in accordance with **Rule A1**), and that the terminal ordering (**Rule A3**) of input terminals and then output terminal, is adhered to.

Finally, the `PERMUTABLE` statement can be analysed thus:

`P_1`	states that the terminals `T_1` and `T_2` (Gate A) are interchangeable.
`P_2`	states that the terminals `T_4` and `T_5` (Gate B) are interchangeable.
`P_3`	states that the terminals `T_9` and `T_10` (Gate C) are interchangeable.
`P_4`	states that the terminals `T_12` and `T_13` (Gate D) are interchangeable.
`P_5`	states that groups `NAND_A` to `NAND_D` (Gates A to D) are interchangeable, so long as the sequence order of the terminals is adhered to.

Note that in the above example, we use "`P_1 = T_1, T_2;`" and not "`P_1 = NAND_INA;`", as the latter statement would violate **Rule B1** in not having at least two elements.

As **Rule B3** states, the ordering of the elements is not important, such that the statement

```
P_5 = NAND_A, NAND_B, NAND_C, NAND_D;
```

shall have the same interpretation as

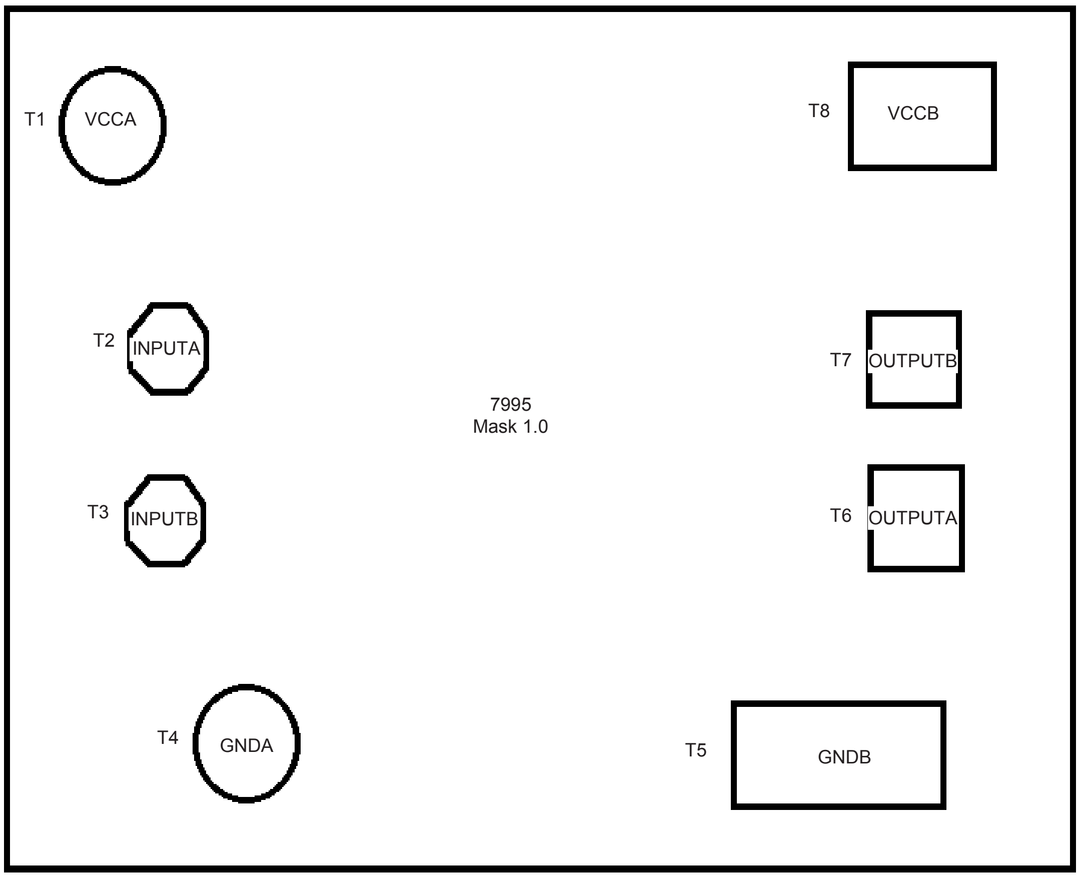
```
P_5 = NAND_D, NAND_C, NAND_B, NAND_A;
```

As **Rule B2** states, the following would **NOT** be acceptable

```
P_7 = NAND_D, T_1, T_2;        (unequal elements, also breaks Rule B1)
P_8 = NAND_INA, NAND_B;        (unequal elements)
P_9 = NAND_INA, NAND_A;        (unequal elements and possible recursion)
P_10 = NAND_A, NAND_A; (possible recursion)
```

Annex C
(informative)

A Typical CAD view from the DDX file block example given in Annex A



IEC 896/11

Figure C.1 – CAD representation of DDX example from Annex A

Note that Figure C.1. is not intended to be a geometrically accurate drawing, the purpose is solely to assist visualise the output from the DDX example given in Annex A. (The Die Fiducial graphic is not shown). The CAD system has chosen to display the **DEVICE_NAME** (refer to 8.1.1) and the **DIE_MASK_REVISION** (refer clause 8.2.3) parameters for the die. The individual terminal identifiers and terminal names have similarly been selected for display (refer to 8.4.5 for details).

Annex D
(informative)

Properties for Simulation

A DDX file can reference other external files, such as electrical simulation model files, mechanical model files and thermal modelling files.

To do this adequately, the following data shall be present:

– the file name and its creation date,
– the exact name of the simulator,
– its version
– the relevant compliance level to which this model applies,
– The terminals to which the model applies (electrical).

Within the model file there should be a textual note relating to the model's capability, applicability and accuracy, specifically noting any shortcomings of the model and its usage. The text "***simulator***" shall be replaced with either the actual or generic name of the simulator used and/or model data. Typical "***simulator"*** names might be Verilog, VHDL, SPICE, ELDO, BSDL, IBIS etc.

8.7.1 **SIMULATOR_*simulator*_MODEL_FILE**
Data presented as a file name. All file names given shall not include any absolute computer drive, computer path reference, or any details that cause the information to become computer or operating system specific or dependent.

8.7.2 **SIMULATOR_*simulator*_MODEL_FILE_DATE**
Data presented as a date, as per ISO 8601.

8.7.3 **SIMULATOR_*simulator*_NAME**
Data presented as text of the actual simulator used to prove the model.

8.7.4 **SIMULATOR_*simulator*_VERSION**
Data presented as text determining the actual version or revision of the simulator against which the model file was proven.

8.7.5 **SIMULATOR_*simulator*_COMPLIANCE**
Data presented as text which reflects the overall compliance of the simulator model, where applicable. "2G6", "VHDL '93", "IEEE 1364-2001" or "IEEE 1076.3-97" are examples of industry recognised compliance levels.

8.7.6 **SIMULATOR_*simulator*_TERM_GROUP**
Determines the terminals or terminal groups to which the simulator applies when the model file only covers a limited range of terminals, otherwise it is assumed that the model file covers the entire device. This parameter therefore permits the restriction of the simulation model to specific terminals or groups of terminals, and by it's absence, implies that the simulation model is applicable to all device terminals.

Example:

```
SIMULATOR_SPICE_MODEL_FILE = "RTBAE2.MOD";
SIMULATOR_SPICE_MODEL_FILE_DATE = "1997-09-17";
SIMULATOR_SPICE_NAME = "pSpice";
```

```
SIMULATOR_SPICE_VERSION = “4.0.1”;
SIMULATOR_SPICE_COMPLIANCE = “2G6”;
TERMINAL_GROUP G_1 = P8, P9, P10;
TERMINAL_GROUP G_2 = P28, P29, P30;
SIMULATOR_SPICE_TERM_GROUP = T_1, T_2, G_1, G_2;
```

Annex E
(informative)

TERMINAL and TERMINAL_TYPE graphical usage for CAD/CAM systems

The creation of a CAD/CAE "view" of the device from the data within the **DEVICE** block may be considered as having four separate steps. There is no geometric layer information contained within the DDX format, and it is assumed that all geometric data shall be displayed on a single layer. Textual information may be provided on a separate layer.

The **GEOMETRIC_UNITS** parameter (see 8.3.1) defines the appropriate scaling factors to be employed, and sets the scene for all further numerical input from that **DEVICE** block. The **GEOMETRIC_VIEW** parameter (see 8.3.2) determines whether the device is being viewed from the top or bottom.

The device outline is created from the **SIZE** parameter (see 8.3.4), and any geometric offset for further geometric placement is calculated from the **GEOMETRIC_ORIGIN** parameter (see 8.3.3). Data such as device name and type is collected and prepared for display purposes. Note that the **GEOMETRIC_ORIGIN** parameter is referenced to the XSIZE and YSIZE parameters, before tolerancing, as XSIZE/2, YSIZE/2. The **GEOMETRIC_ORIGIN** parameter values are added to this reference for all relative geometric parameters. The **SIZE_TOLERANCE** parameter (see 8.3.5) values may also be used to indicate the minimum and maximum device outline.

The **TERMINAL_COUNT**, **TERMINAL_TYPE_COUNT** and **CONNECTION_COUNT** parameters (see 8.4.1, 8.4.2 and 8.4.3) are essentially duplicated, as their values may be calculated from the **TERMINAL** (see 8.4.5) and **TERMINAL_TYPE** (see 8.4.4) structures. They are intended, however, to be used as a "sanity" check, to ensure correct parsing, and highlight any file or BLOCK corruption. The **TERMINAL_TYPE_COUNT** value can only be less than or equal to the **TERMINAL_TYPE** value. The **CONNECTION_COUNT** value has no such restriction.

The shapes of the terminals or connecting areas are separately defined, each shape having its own geometric reference centre. These shapes are defined with the **TERMINAL_TYPE** structure. Each of these terminal shapes can be placed anywhere within the device outline and, as there is no placement limitation, may be placed outside the device outline should the user so desire (e.g. pad placement for PCB-type connections). The **TERMINAL_TYPE** structure introduces **Terminal_Type_Names** (see 8.4.4.1 and 8.4.5.3) that shall be unique within the **DEVICE** block; these names are placeholders for subsequent usage by the **TERMINAL** structure, and so should be introduced prior to the **TERMINAL** structure statement using the placeholder name to permit single pass parsing of the **DEVICE** block. To prevent any possible ambiguity, it is strongly advised that the **Terminal_Type_Names** be limited to alphanumeric characters.

These terminals types (shapes), having been defined, may now be referenced by their **Terminal_Type_Name**, and geometrically placed by the **TERMINAL** structure in sequence (***T_n***), at given locations (the X & Y co-ordinates), orientated by mirroring (Figure E.1) and clockwise rotation (Figure E.2), and finally assigned a terminal name and I/O function for further use by the CAD system. The **Terminal_name** (see 8.4.5.7) would usually be visible as text, for visual convenience and need not be unique. The **Terminal_name** can also indicate terminals that will usually be electrically connected together, such as *"VDD", "VSS", "Ground"* etc.. A connection number (***conn_N***) may also be assigned and whilst this number bears no relation to the actual terminal number, it can be used for identifying common groups of pins, (such as pins that must be connected together), or actually referring to the original device's packaged pin-out (which again can be different to the die pad numbering). The functional I/O details (***IO_type***) (see 8.4.5.8) of each terminal are available for further connectivity checks by the CAD system. The unique **TERMINAL** identifier, (***T_n)*** (see 8.4.5.1)***,*** can have any mix of alphanumeric characters, and need not be in, nor follow, any particular sequence. The only requisite is that, as an identifier, it is unique within that **DEVICE** block, and to prevent any possible ambiguity, it is strongly advised that the unique identifier, (***T_n)*** be limited to alphanumeric characters.

Where applicable, the **DIE_SUBSTRATE_CONNECTION** (see 8.5.5) details should be displayed, alerting the user to the need to facilitate appropriate connection.

The creation and placement of fiducials is identical to that for terminals by using the **FIDUCIAL_TYPE** (see 8.3.8) and **FIDUCIAL** (see 8.3.9) parameter structures (q.v.). The main differences being that

- graphical fiducial data are only held within a separate graphic file, and
- that fiducials do not have any connectivity or electrical properties.

It is up to the CAD/CAM system to determine as to how to display the graphic content of the fiducial graphic file, as there are currently a plethora of graphical file standards. The fiducial graphic shall be scaled to fit within the size dimensions given by the **FIDUCIAL_TYPE** parameter structure, whose reference and placement centre will be taken as the midpoint of the X- and Y-size dimensions. The actual content and type of graphical data within this file is commonly determined by the file extent. i.e. TIFF, GIF, JPEG, BMP.

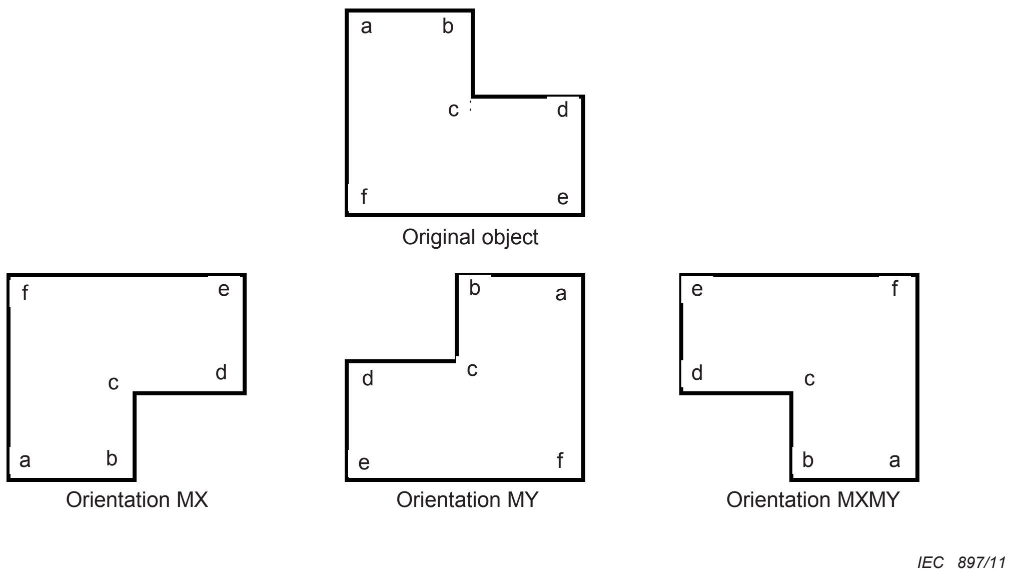


Figure E.1 – Highlighting the MX and MY orientation properties

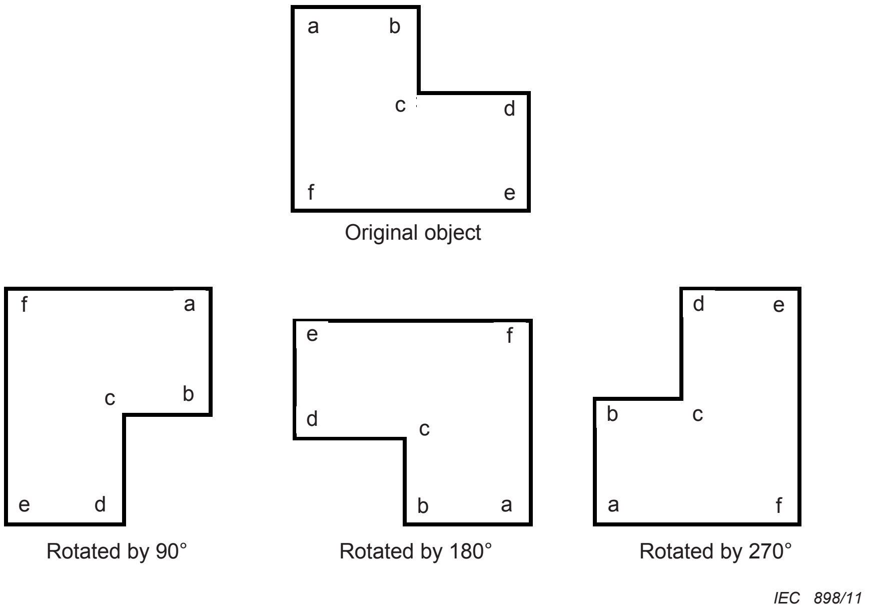


Figure E.2 – Highlighting the angular rotational orientation properties

Annex F
(informative)

Cross-reference with IEC 61360-4

The table below provides a read-across between the parameters in this standard and the DET definitions given in IEC 61360-4:2005.

Table F.1 – Parameter List

Clause/ Sub-clause	Parameter Name	Parameter Type	Data Type	DET Code
8.1	***BLOCK DATA***			
8.1.1	DEVICE_NAME	Header	Text	AAH547-001
8.1.2	DEVICE_FORM	Header	Text	AAD004-001
8.1.3	BLOCK_VERSION	Variable	Text	
8.1.4	BLOCK_CREATION_DATE	Variable	Date	
8.1.5	VERSION	Variable	Text	
8.2	***DEVICE_DATA***			
8.2.1	DIE_NAME	Variable	Text	AAD002-001
8.2.2	DIE_PACKAGED_PART_NAME	Variable	Text	AAD143-001
8.2.3	DIE_MASK_REVISION	Variable	Text	AAD003-001
8.2.4	MANUFACTURER	Variable	Text	AAD140-001
8.2.5	DATA_SOURCE	Variable	Text	AAD142-001
8.2.6	DATA_VERSION	Variable	Text	
8.2.7	FUNCTION	Variable	Text	
8.2.8	IC_TECHNOLOGY	Variable	Text	AAE686-005
8.2.8	DEVICE_PICTURE_FILE	Variable	File Name	
8.2.9	DEVICE_DATA_FILE	Variable	File Name	
8.3	***GEOMETRIC DATA***			
8.3.1	GEOMETRIC_UNITS	Variable	Real	AAD115-001
8.3.2	GEOMETRIC_VIEW	Variable	Text	AAD144-001
8.3.3	GEOMETRIC_ORIGIN	Variable	Real	AAD129 & 130-001
8.3.4	SIZE	Variable	Real	AAD070 & 071-001
8.3.5	SIZE_TOLERANCE	Variable	Real	AAD117-001
8.3.6	THICKNESS	Variable	Real	AAD072-001
8.3.7	THICKNESS_TOLERANCE	Variable	Real	AAD118-001
8.3.8	FIDUCIAL_TYPE	Structure		AAD156 to 159-001
8.3.9	FIDUCIAL	Structure		AAD160 to 162-001
8.4	***TERMINAL DATA***			
8.4.1	TERMINAL_COUNT	Variable	Integer	AAD145-001
8.4.2	TERMINAL_TYPE_COUNT	Variable	Integer	AAD116-001
8.4.3	CONNECTION_COUNT	Variable	Integer	
8.4.4	TERMINAL_TYPE	Structure		AAD024 to 030, AAD121-001

Clause/ Sub-clause	Parameter Name	Parameter Type	Data Type	DET Code
8.4.5	TERMINAL	Structure		AAD014 to 016, AAD019, AAD020-001
8.4.6	TERMINAL_GROUP	Structure		
8.4.7	PERMUTABLE	Structure		
8.5	***MATERIAL DATA***			
8.5.1	TERMINAL_MATERIAL (was DIE_TERMINAL_MATERIAL)	Variable	Text	AAD120-001, AAE634-005
8.5.2	TERMINAL_MATERIAL_STRUCTURE	Variable	Text	
8.5.3	DIE_SEMICONDUCTOR_MATERIAL	Variable	Text	AAD148-001
8.5.4	DIE_SUBSTRATE_MATERIAL	Variable	Text	AAD005-001
8.5.5	DIE_SUBSTRATE_CONNECTION	Variable	Text	AAD006 & 007-001
8.5.6	DIE_PASSIVATION_MATERIAL	Variable	Text	AAD078-001
8.5.7	DIE_BACK_DETAIL	Variable	Text	AAD119-001
8.6	***ELECTRICAL and THERMAL DATA***	Variable		
8.6.1	MAX_TEMP	Variable	Real	AAD149-001
8.6.2	MAX_TEMP_TIME	Variable	Real	
8.6.3	POWER_RANGE	Variable	Real	AAD151-001
8.6.4	TEMPERATURE_RANGE	Variable	Real	AAE891-005
8.7	***SIMULATOR DATA***			
8.7.1	SIMULATOR_*simulator*_MODEL_FILE	Variable	File Name	
8.7.2	SIMULATOR_*simulator*_MODEL_FILE_DATE	Variable	Text	
8.7.3	SIMULATOR_*simulator*_NAME	Variable	Text	
8.7.4	SIMULATOR_*simulator*_VERSION	Variable	Text	
8.7.5	SIMULATOR_*simulator*_COMPLIANCE	Variable	Text	
8.7.6	SIMULATOR_*simulator*_TERM_GROUP	Variable		
8.8	***HANDLING, PACKING, STORAGE and ASSEMBLY***			
8.8.1	DELIVERY_FORM (was DIE_DELIVERY_FORM)	Variable	Text	AAD155-001
8.8.2	PACKING_CODE	Variable	Text	AAD055-001
8.8.8	ASSEMBLY parameters	Variable	Text	
8.9	***WAFER SPECIFIC DATA***			
8.9.1	WAFER_SIZE	Variable	Text	AAD011-001
8.9.2	WAFER_THICKNESS	Variable	Real	
8.9.3	WAFER_THICKNESS_TOLERANCE	Variable	Real	
8.9.4	WAFER_DIE_STEP_SIZE	Variable	Real	AAD163-001, AAD164-001
8.9.5	WAFER_GROSS_DIE_COUNT	Variable	Integer	AAD165-001
8.9.6	WAFER_INDEX	Variable	Real	AAD166-001, AAD167-001
8.9.7	WAFER_RETICULE_STEP_SIZE	Variable	Real	AAD168-001, AAD169-001
8.9.8	WAFER_RETICULE_GROSS_DIE_COUNT	Variable	Integer	AAD170-001

Clause/ Sub-clause	Parameter Name	Parameter Type	Data Type	DET Code
8.9.9	WAFER_INK PARAMETERS	Variable	Text	
8.10	***BUMP TERMINATION SPECIFIC DATA***			
8.10.1	BUMP_MATERIAL	Variable	Text	AAD124-001
8.10.2	BUMP_HEIGHT	Variable	Real	AAD146-001
8.10.3	BUMP_HEIGHT_TOLERANCE	Variable	Real	AAD147-001
8.10.4	BUMP_SHAPE	Variable	Text	
8.10.5	BUMP_SIZE	Variable	Real	
8.10.6	BUMP_SPECIFICATION_DRAWING	Variable	File Name	
8.10.7	BUMP_ATTACHMENT_METHOD	Variable	Text	
8.11	***MINIMALLY PACKAGE DEVICE SPECIFIC DATA***			
8.11.1	MPD_PACKAGE_MATERIAL	Variable	Text	AAD150-001
8.11.2	MPD_PACKAGE_STYLE	Variable	Text	
8.11.3	MPD_CONNECTION_TYPE	Variable	Text	
8.11.4	MPD_MSL_LEVEL	Variable	Text	
8.11.5	MPD_PACKAGE_DRAWING	Variable	File Name	
Deleted	MPD_DELIVERY_FORM (refer to 8.8.1)	Variable		
Deleted	MPD_CONNECTION_MATERIAL (refer to 8.5.1)	Variable		
8.12	***QUALITY, RELIABILITY and TEST DATA***			
8.12.1	QUALITY Parameters	Variable	Text	
8.12.1	TEST Parameters	Variable	Text	
8.13	***OTHER DATA***			
8.13.1	TEXT	Variable	Text	

Annex G
(informative)

Notes on VERSION and NAME parameters

There are three “VERSION” related parameters, each of which contains specific revision data concerned with the different aspects of the data presented within a DDX BLOCK.

VERSION Subclause 8.1.5
This relates to the IEC 62258, Part 2, and its issue. Being a data transfer standard for use with CAD/CAE systems, it is anticipated that there will be updates and revisions, particularly the inclusion of additional data, on a regular basis. Refer to Clause 1 to determine the DDX version covered in this standard.

BLOCK_VERSION Subclause 8.1.3
This parameter relates solely to the revision status of the data with the BLOCK, and would expect to be “up-revved” as data within the BLOCK is modified. The **BLOCK_VERSION** parameter is specifically NOT concerned with the source of any data changes.

DATA_VERSION Subclause 8.2.6
This parameter covers the source of the data used; to indicate the technical “state-of-the-art” for the data contents. This parameter is therefore closely coupled with the **DATA_SOURCE** parameter.

BLOCK_CREATION_DATE Subclause 8.1.4
This parameter relates solely to the last date that data within the block was revised, and always accompanies, and should be linked to, the **BLOCK_VERSION** parameter.
Likewise, there are four “NAME” related parameters, each of which represents a different aspect of the name by which the device is known.

DEVICE_NAME Subclause 8.1.1
This is often referred to as the type number and is the identity normally used in data sheets and in parts lists.

DIE_NAME Subclause 8.2.1
This is the name given to the die itself and the set of masks used in its fabrication. This name is often part of the fiducials on the die surface.

DIE_MASK_REVISION Subclause 8.2.3
The revision or version code of the mask set used in fabricating the die.

DIE_PACKAGED_PART_NAME Subclause 8.2.2
This is generally similar to the DEVICE_NAME and is quoted as the name of a packaged part which usually contains the same die and has similar electrical characteristics.

Annex H
(informative)

Notes on WAFER parameters

When die are delivered as unsawn wafers, additional parameters can assist in the practicality of mechanical handling and quantity purchasing. Such parameters include the die step size, wafer size, wafer flat angle and gross die per wafer. Where the die step size is irregular, often caused by the reticule step, further parameters relating to the reticule size can become important.

Parameters relating to wafer delivery include:

WAFER_SIZE Subclause 8.9.1
This parameter gives the approximate wafer diameter. This is commonly quoted in millimetres or inches, and so need not be related to the **GEOMETRIC_UNITS** parameter.

WAFER_DIE_STEP_SIZE Subclause 8.9.4
The X and Y co-ordinates specified by this parameter are commonly used for die sawing and wafer probing, etc. The units again are determined by the **GEOMETRIC_UNITS** parameter.

WAFER_GROSS_DIE_COUNT Subclause 8.9.5
The gross die count can be an important parameter in the calculation of nett die per wafer, but does not reflect the good die per wafer, only the total of viable relevant die (whole die). Whilst theoretically there is a direct mathematical relationship between wafer and die size, test “drop-ins” etc., will affect the final count. The user should also be aware of cost-saving techniques which involve the reduction in the number of masks required that often result in fewer viable die.

WAFER_INDEX Subclause 8.9.6
This parameter pair defines the both type of physical feature present on a wafer and its orientation, which may then be used in an auto-location system in order to determine the placement and orientation of the wafer.
The first parameter specifies the type of feature, either a “flat” or “notch”, whilst the second parameter determines the approximate angle, to the nearest degree, of this index feature on the wafer with respect to the X-axis on the surface of the die. This can assist in wafer location recognition system for initial or crude orientation prior to final alignment by optical means. Where more than one flat exists on the wafer, the orientation is with respect to the major, or prime, flat, which may also be used to indicate the wafer lattice crystal [110] direction.

WAFER_RETICULE_STEP_SIZE Subclause 8.9.7
Where the reticule dimensions interfere with the regular stepping of the die size, the reticule step size should be given, in the same dimensions as the die step size (**GEOMETRIC_UNITS**).

WAFER_RETICULE_GROSS_DIE_COUNT Subclause 8.9.8
Where the **WAFER_RETICULE_STEP_SIZE** parameter is relevant, it may also be relevant to state the number of die, of the specified die type, within the reticule window. In the case of MPW runs, this may only be one die.

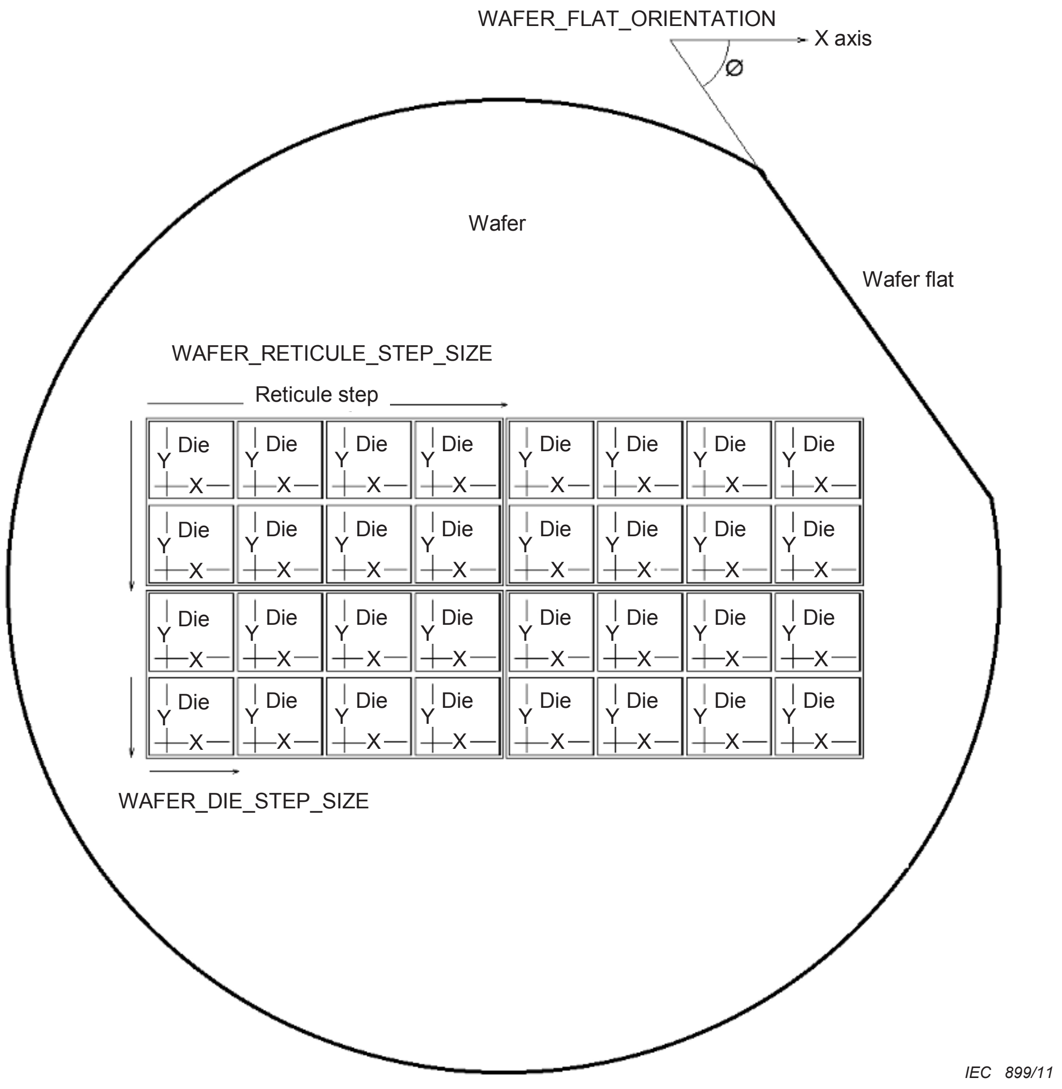


Figure H.1 – Illustrating the WAFER parameters

Figure H.1 shows a representation of where the reticule step size interferes with the die step dimensions, (the reticule here containing a total of 8 die in a 4 by 2 matrix). The gap between reticule and reticule results in an additional distance between adjacent die at the boundaries of the reticule. Figure H.1 also shows then wafer flat angle with respect to the assumed X-dimension of the die. Notional parameter values for Figure H.1 might be:

```
WAFER_SIZE = "150mm";
WAFER_DIE_STEP_SIZE = 1000,1000;
WAFER_GROSS_DIE_COUNT = 2077;
WAFER_INDEX = "Flat",45;
WAFER_RETICULE_STEP_SIZE = 4050, 2100;
WAFER_RETICULE_GROSS_DIE_COUNT = 8;
```

The reticule parameters given in this example here demonstrates a 50 unit discrepancy (**GEOMETRIC_UNITS)** in the X-dimension, and a 100 unit discrepancy (**GEOMETRIC_UNITS)** in the Y-dimension.

Annex I
(informative)

Additional notes

1) As it stands, the DDX format requires data to reside within a block structure, strictly delimited by braces. The only data required outside of these braces are the **DEVICE** keyword and the associated parameters of **DEVICE_NAME** and **DEVICE_FORM**. This then leaves scope for future expansion, such as the inclusion of non-DDX format data outside of the DDX block structure, provided that the DDX structure and format rules are conformed to. Such additional data may include file and block checksums to validate the data file contents, although full ASCII character checksum calculations need to take into account the differing line delimiters used by different software operating systems. This additional data should be catered for, and ignored, by any parsing routine specifically for reading DDX formatted data.

2) As there are many different character conventions for displaying or annotating a negated signal, such as “\”, “/”, “N”, “B” or “not/bar”, no particular method has been adopted in the DDX format for the display of such signal names. It is therefore reliant upon the information provider to adopt a single convention, and the CAE vendor to accommodate this convention in the graphical display, where required.

3) Referencing an external file. File name references are required by:

8.3.8.2	**FIDUCIAL_TYPE**
8.2.9	**DEVICE_PICTURE_FILE**
8.2.10	**DEVICE_DATA_FILE**
8.11.5	**MPD_PACKAGE_DRAWING**
8.10.6	**BUMP_SPECIFICATION_DRAWING**

When referencing an external file, such as a graphic, textual or document file, it is preferable that the external file type conforms to a standard format in common use. Convention holds that the document format and/or type is indicated by the named file extent.

4) Multiple file names. Multiple file names are permitted in:

8.2.9	**DEVICE_PICTURE_FILE**
8.2.10	**DEVICE_DATA_FILE**
8.11.5	**MPD_PACKAGE_DRAWING**
8.10.6	**BUMP_SPECIFICATION_DRAWING**

Where multiple file names are permitted, the file names can be introduced in a single function invocation, as:

```
DEVICE_DATA_FILE = “file1.txt”, “file2.txt”, … , “fileX.txt”;
```

or as multiple invocations of that function, as:

```
DEVICE_DATA_FILE = “file1.txt”;
DEVICE_DATA_FILE = “file2.txt”;
.
DEVICE_DATA_FILE = “fileN.txt”;
```

When using a single function invocation to introduce more than one file name, each file name should be surrounded by double quotes (as 6.3.7) and each file name shall be separated by a comma (as 6.3.2). Each line shall be terminated by a semicolon as 6.3.1.

Annex J
(informative)

DDX Version history

The following table, Table J.1, indicates the parameter version history. The issue column reflects the current DDX revision and version number at which that parameter was either introduced or last altered.

The current version for this DDX format is 1.3.0, as stated in Clause 1 of this standard.

The previous version of this DDX format was 1.0, as defined in ES59008, Part 6-1.

Table J.1 – Parameter Change History List

Clause/ Subclau-se	Parameter Name	ES59008-6-1 Reference	Previous Clause	Current Issue
8.1	***BLOCK DATA***			
8.1.1	`DEVICE_NAME`	8.2	8.5	1.0
8.1.2	`DEVICE_FORM`	8.1	8.4	1.0
8.1.3	`BLOCK_VERSION`	8.17	8.2	1.0
8.1.4	`BLOCK_CREATION_DATE`	8.16	8.1	1.0
8.1.5	`VERSION`	8.3	8.3	1.0
8.2	***DEVICE_DATA***			
8.2.1	`DIE_NAME`		8.8	1.2.1
8.2.2	`DIE_PACKAGED_PART_NAME`	8.36	8.9	1.0
8.2.3	`DIE_MASK_REVISION`	8.30	8.6	1.0
8.2.4	`MANUFACTURER`	8.4	8.7	1.0
8.2.5	`DATA_SOURCE`	8.19	8.11	1.0
8.2.6	`DATA_VERSION`	8.55	8.12	1.0
8.2.7	`FUNCTION`	8.5	8.10	1.0
8.2.8	`IC_TECHNOLOGY`	8.20	8.25	1.0
8.2.8	`DEVICE_PICTURE_FILE`			1.3.0
8.2.9	`DEVICE_DATA_FILE`			1.3.0
8.3	***GEOMETRIC DATA***			
8.3.1	`GEOMETRIC_UNITS`	8.6	8.13	1.0
8.3.2	`GEOMETRIC_VIEW`	8.7	8.14	1.0
8.3.3	`GEOMETRIC_ORIGIN`	8.10	8.17	1.0
8.3.4	`SIZE`	8.8	8.15	1.0
8.3.5	`SIZE_TOLERANCE`	8.21	8.18	1.0
8.3.6	`THICKNESS`	8.9	8.16	1.0
8.3.7	`THICKNESS_TOLERANCE`	8.22	8.19	1.0
8.3.8	`FIDUCIAL_TYPE`	8.47	8.51	1.0
8.3.9	`FIDUCIAL`	8.48	8.52	1.0
8.4	***TERMINAL DATA***			

Clause/ Subclau-se	Parameter Name	ES59008-6-1 Reference	Previous Clause	Current Issue
8.4.1	TERMINAL_COUNT	8.11	8.20	1.0
8.4.2	TERMINAL_TYPE_COUNT	8.12	8.21	1.0
8.4.3	CONNECTION_COUNT	8.13	8.22	1.0
8.4.4	TERMINAL_TYPE	8.14	8.23	1.0
8.4.5	TERMINAL	8.15	8.24	1.0
8.4.6	TERMINAL_GROUP		8.58	1.3.0
8.4.7	PERMUTABLE		8.61	1.3.0
8.5	***MATERIAL DATA***			
8.5.1	TERMINAL_MATERIAL (was DIE_TERMINAL_MATERIAL)		8.30	1.3.0
8.5.2	TERMINAL_MATERIAL_STRUCTURE			1.3.0
8.5.3	DIE_SEMICONDUCTOR_MATERIAL	8.34	8.26	1.0
8.5.4	DIE_SUBSTRATE_MATERIAL	8.32	8.27	1.0
8.5.5	DIE_SUBSTRATE_CONNECTION	8.31	8.28	1.0
8.5.6	DIE_PASSIVATION_MATERIAL	8.37	8.29	1.0
8.5.7	DIE_BACK_DETAIL	8.33	8.31	1.0
8.6	***ELECTRICAL and THERMAL DATA***			
8.6.1	MAX_TEMP	8.18	8.33	1.0
8.6.2	MAX_TEMP_TIME			1.3.0
8.6.3	POWER_RANGE	8.23	8.34	1.0
8.6.4	TEMPERATURE_RANGE	8.24	8.35	1.0
8.7	***SIMULATOR DATA***			
8.7.1	SIMULATOR_*simulator*_MODEL_FILE	8.25	8.36	1.0
8.7.2	SIMULATOR_*simulator*_MODEL_FILE_DATE	8.26	8.37	1.0
8.7.3	SIMULATOR_*simulator*_NAME	8.27	8.38	1.0
8.7.4	SIMULATOR_*simulator*_VERSION	8.28	8.39	1.0
8.7.5	SIMULATOR_*simulator*_COMPLIANCE	8.29	8.40	1.0
8.7.6	SIMULATOR_*simulator*_TERM_GROUP		8.59	1.3.0
8.8	***HANDLING, PACKING, STORAGE and ASSEMBLY***			
8.8.1	DELIVERY_FORM (was DIE_DELIVERY_FORM)	8.35	8.41	1.0
8.8.2	PACKING_CODE	8.46	8.42	1.0
8.8.8	ASSEMBLY parameters			1.3.0
8.9	***WAFER SPECIFIC DATA***			
8.9.1	WAFER_SIZE	8.45	8.32	1.0
8.9.2	WAFER_THICKNESS			1.3.0
8.9.3	WAFER_THICKNESS_TOLERANCE			1.3.0
8.9.4	WAFER_DIE_STEP_SIZE		8.53	1.2.1
8.9.5	WAFER_GROSS_DIE_COUNT		8.54	1.2.1
8.9.6	WAFER_INDEX		8.55	1.2.1
8.9.7	WAFER_RETICULE_STEP_SIZE		8.56	1.2.1
8.9.8	WAFER_RETICULE_GROSS_DIE_COUNT		8.57	1.2.1

Clause/ Subclau-se	Parameter Name	ES59008-6-1 Reference	Previous Clause	Current Issue
8.9.9	WAFER_INK			1.3.0
8.10	***BUMP TERMINATION SPECIFIC DATA***			
8.10.1	BUMP_MATERIAL	8.38	8.43	1.0
8.10.2	BUMP_HEIGHT	8.39	8.44	1.0
8.10.3	BUMP_HEIGHT_TOLERANCE	8.40	8.45	1.0
8.10.4	BUMP_SHAPE			1.3.0
8.10.5	BUMP_SIZE			1.3.0
8.10.6	BUMP_SPECIFICATION_DRAWING			1.3.0
8.10.7	BUMP_ATTACHMENT_METHOD			1.3.0
8.11	***MINIMALLY PACKAGE DEVICE SPECIFIC DATA***			
8.11.1	MPD_PACKAGE_MATERIAL	8.41	8.46	1.0
8.11.2	MPD_PACKAGE_STYLE		8.47	1.2.1
8.11.3	MPD_CONNECTION_TYPE	8.43	8.49	1.0
8.11.4	MPD_MSL_LEVEL			1.30
8.11.5	MPD_PACKAGE_DRAWING			1.30
Deleted	MPD_DELIVERY_FORM (refer to 8.8.1)	8.42	8.48	1.0
Deleted	MPD_CONNECTION_MATERIAL (refer to 8.5.1)	8.44	8.50	1.0
8.12	***QUALITY, RELIABILITY and TEST DATA***			
8.12.1	QUALITY Parameters			1.3.0
8.12.1	TEST Parameters	.		1.3.0
8.13	***OTHER DATA***			
8.13.1	TEXT		8.60	1.3.0
8.14	***CONTROL DATA***			
8.14.1	PARSE			1.3.0

Annex K
(informative)

Parse Control

The PARSE_ series of parameters are intended to add in-line control of the parsing software, primarily to allow for additions and extensions to the DDX software that have not, to date, been ratified and included in the current standard. Note that none of the PARSE_ parameters have any relevance or meaning to the device data.

To this end a number of PARSE_ parameters have been added:

8.14.1	**PARSE_MODE**
8.14.2	**PARSE_ERROR_REPORT**
8.14.3	**PARSE_ERROR_TRAP**
8.14.4	**PARSE_IGNORE**
8.14.5	**PARSE_DEFINE_PARAMETER**
8.14.6	**PARSE_DEFINE_STRUCTURE**

The **PARSE_MODE**, **PARSE_ERROR_REPORT**, **PARSE_ERROR_TRAP** and **PARSE_IGNORE** parameters may have multiple occurrences, and are intended to "switch" the desired parse mode as the DDX file or DDX block is linearly read.

The **PARSE_DEFINE_PARAMETER** and **PARSE_DEFINE_STRUCTURE** parameters serve to introduce new parameters and/or structures to the parsing software for inclusion in syntax checks, data validation and value assignment, solely for that specific DDX block. Once defined, it is expected that the new parameters and structures will fully conform to the syntax etc., as defined in Clauses 5 to 7. Until the new parameters and structures have been ratified and included in an update to this DDX standard, all data shall be treated as textual and will have no associated S.I unit.

The **PARSE_MODE** parameter is intended to control the way in which the syntax and data rules are interpreted and applied. In all cases, data that complies with this version of the DDX format (refer Clause 1) shall be free from errors or warnings. New data parameters or structures that have been introduced using the **PARSE_DEFINE**_xxx parameters may require some rule relaxation before an error-free parse result is achieved.

```
PARSE_MODE = STRICT;
```
Under "normal" circumstances, the _MODE parameter will default to "STRICT", and all currently compliant data must be error free, and errors will be issued if the PARSE_DEFINE options are used. These errors are to or structures than those given in the current standard,

```
PARSE_MODE = RELAXED;
```
The "RELAXED" value is given to permit inclusion of additional defined data that may not strictly comply with the rules as written. Instead of errors, warnings are to be issued when new parameters are included to serve notice to the user that the DDX block under scrutiny contains parameters not currently in the standard.

```
PARSE_MODE = USER;
PARSE_MODE = ENHANCED;
```
The "ENHANCED" and "USER" values are alternative options available to the discretion of the parse software provider and user.

The **PARSE_ERROR_REPORT** parameter determines the level of warnings and errors reported DURING parsing, it should have no effect on the end report. This is primarily to prevent warnings regarding parameters being used prior to their assignment, caused by the introduction of new parameters. These parameter options are self-explanatory.

```
PARSE_ERROR_REPORT = OFF;
PARSE_ERROR_REPORT = TERSE;
PARSE_ERROR_REPORT = VERBOSE;
```

The **PARSE_ERROR_TRAP** parameter determines how the parsing software should react once an error (as opposed to a warning) is discovered.

```
PARSE_ERROR_TRAP = ALL;
```

This is the expected "default" mode. The DDX block is parsed and all errors are reported.

```
PARSE_ERROR_TRAP = FIRST;
```

The "FIRST" value should cause the parsing software to halt upon the detection of the first error.

The **PARSE_IGNORE** parameter is meant to control whether checking is actually performed of part of the DDX block or not, and should accordingly be used with care.

```
PARSE_IGNORE = NONE;
PARSE_IGNORE = OFF;
```

All parsing checks are performed. This should be the default state for the parsing software.

```
PARSE_IGNORE = ALL;
```

All parsing checks are ignored, this is useful only to bypass rubbish data

```
PARSE_IGNORE = SYNTAX_ONLY;
```

Only syntactical errors and line-termination checks are performed. There are no checks on forward/backward referencing of variables, nor of parameter counts etc. This option should only be used when new parameters or structures are introduced that include variable referencing or non-textual data.

WARNINGS and ERRORS

It is advised that the following items be classed as **WARNINGS**, and all other errors classified as **ERRORS**:

WARNING 1.	The occurrence of ASCII characters in the range 0x80 to 0xFF
WARNING 2.	Where a data line is assumed to be terminated due to its length exceeding the parser's line input buffer. Refer to 6.3.9.
WARNING 3.	Where textual data, not enclosed within double quotes, includes line break characters, refer to 6.3.8.
WARNING 4.	Where the textual file name does not conform to the character set as defined in 7.1.3.2.
WARNING 5.	Where the **PARSE_DEFINE**_xxx introduces a parameter or structure that has been ratified and defined within the later versions of the DDX standard.

This page deliberately left blank

British Standards Institution (BSI)

BSI is the national body responsible for preparing British Standards and other standards-related publications, information and services.

BSI is incorporated by Royal Charter. British Standards and other standardization products are published by BSI Standards Limited.

About us

We bring together business, industry, government, consumers, innovators and others to shape their combined experience and expertise into standards -based solutions.

The knowledge embodied in our standards has been carefully assembled in a dependable format and refined through our open consultation process. Organizations of all sizes and across all sectors choose standards to help them achieve their goals.

Information on standards

We can provide you with the knowledge that your organization needs to succeed. Find out more about British Standards by visiting our website at bsigroup.com/standards or contacting our Customer Services team or Knowledge Centre.

Buying standards

You can buy and download PDF versions of BSI publications, including British and adopted European and international standards, through our website at bsigroup.com/shop, where hard copies can also be purchased.

If you need international and foreign standards from other Standards Development Organizations, hard copies can be ordered from our Customer Services team.

Subscriptions

Our range of subscription services are designed to make using standards easier for you. For further information on our subscription products go to bsigroup.com/subscriptions.

With **British Standards Online (BSOL)** you'll have instant access to over 55,000 British and adopted European and international standards from your desktop. It's available 24/7 and is refreshed daily so you'll always be up to date.

You can keep in touch with standards developments and receive substantial discounts on the purchase price of standards, both in single copy and subscription format, by becoming a **BSI Subscribing Member**.

PLUS is an updating service exclusive to BSI Subscribing Members. You will automatically receive the latest hard copy of your standards when they're revised or replaced.

To find out more about becoming a BSI Subscribing Member and the benefits of membership, please visit bsigroup.com/shop.

With a **Multi-User Network Licence (MUNL)** you are able to host standards publications on your intranet. Licences can cover as few or as many users as you wish. With updates supplied as soon as they're available, you can be sure your documentation is current. For further information, email bsmusales@bsigroup.com.

Revisions

Our British Standards and other publications are updated by amendment or revision.

We continually improve the quality of our products and services to benefit your business. If you find an inaccuracy or ambiguity within a British Standard or other BSI publication please inform the Knowledge Centre.

Copyright

Useful Contacts:

Customer Services
Tel: +44 845 086 9001
Email (orders): orders@bsigroup.com
Email (enquiries): cservices@bsigroup.com

Subscriptions
Tel: +44 845 086 9001
Email: subscriptions@bsigroup.com

Knowledge Centre
Tel: +44 20 8996 7004
Email: knowledgecentre@bsigroup.com

Copyright & Licensing
Tel: +44 20 8996 7070
Email: copyright@bsigroup.com

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK

bsi.

...making excellence a habit.™