INTERNATIONAL STANDARD

ISO/IEC 14568

First edition
1997-05-15

Information technology — DXL: Diagram eXchange Language for tree-structured charts

Technologies de l'information — DXL: Langage pour échange de diagramme pour cartes avec arborescence

INTERNATIONAL ISO/IEC

ISO IEC

Reference number
ISO/IEC 14568:1997(E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organizations to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 14568 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software engineering*.

Annexes A to C of this International Standard are for information only.

Introduction

This International Standard defines DXL (Diagram eXchange Language for tree-structured charts). The purpose of DXL is to facilitate the interchange of different tree-structured charts among CASE tools.

Tree-structured charts and their supporting CASE tools are widely used in algorithm design of software, but their notation is not standardized yet, although Program Constructs were standardized in ISO/IEC 8631. Having different kinds of notation for tree-structured charts causes trouble in large-scale software development: developers are forced to understand unfamiliar notation and sometimes make mistakes in reviewing a design document if the notation is not uniform.

However, it would take a long time to establish and popularize the standard notation, because it would be time consuming and expensive to re-educate designers and modify existing CASE tools to be conformed to the standard. Therefore, it is better to standardize a data exchange language among CASE tools, because:

1. developers can easily read charts in a familiar notation if unfamiliar notation can be converted through the data exchange language; and

2. existing CASE data can also be reused if it can be converted through the data exchange language.

Information technology — DXL: Diagram eXchange Language for tree-structured charts

1 Scope

This International Standard specifies the semantics and syntax of DXL. DXL is a language for exchanging tree-structured charts among CASE tools.

DXL is applicable to:

1. exchanging ISO/IEC 8631 compliant tree-structured charts (examples of which are shown in annex A (informative) of ISO/IEC 8631);
2. exchanging program flowcharts defined in ISO/IEC 5807 if they are well-structured and don't have data defined in ISO/IEC 5807; and
3. describing procedure oriented algorithms.

This International Standard does not specify:

1. graphical information about a chart, such as the shape, size, and location of symbols;
2. configuration information of a chart, such as its version, author, and file name; or
3. information about the data used in the algorithm described by DXL, such as its structure, reading and writing, and declaration.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 646:1991, *Information technology — ISO 7-bit coded character set for information interchange.*

ISO/IEC 2022:1994, *Information technology — Character code structure and extension techniques.*

ISO/IEC 4873:1991, *Information technology — ISO 8-bit code for information interchange — Structure and rules for implementation.*

ISO 5807:1985, *Information processing — Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts.*

ISO/IEC 8631:1989, *Information technology — Program constructs and conventions for their representation.*

ISO 8859-1:1987[1], *Information processing — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1.*

ISO/IEC 10646-1:1993, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.*

[1] Currently under revision.

3 Definition and acronym

For the purposes of this International Standard, the following definition and acronym apply.

3.1 Definition

3.1.1 tree-structured chart: A chart depicting program constructs defined in ISO/IEC 8631 and having the structure of a tree.

3.2 Acronym

3.2.1 CASE: Computer Aided Software Engineering

4 Notation of DXL syntax

The metalanguage used in this International Standard to specify the syntax of the constructs is based on Backus-Naur Form. The notation has been modified from the original to permit more convenient description. Table 1 lists the meanings of the various meta-symbols.

Table 1: Metalanguage Symbols

Metasymbol	Meaning
::=	is defined to be
|	alternatively
[A]	0 or 1 instance of A
{ A }	0 or more instances of A
'XYZ'	terminal symbol XYZ of level 0
''XYZ''	terminal symbol XYZ of level 1
<XYZ>	nonterminal symbol XYZ of level 0
<<XYZ >>	nonterminal symbol XYZ of level 1

There are two levels of compliance, level 0 and level 1. Level 0 includes syntactic elements compliant only to Program Constructs of ISO/IEC 8631. Level 1 includes level 0 and syntactic elements corresponding to subdivided elements of Program Constructs.

5 Definition of DXL

5.1 Lexical elements

```
<character> ::= <one_byte_code_character> | <multi_byte_code_character>
<string> ::= { <character> }
<lexical token> ::= <special-symbol> | <word-symbol> | <identifier> | < specification>
                  | <description> | <comment> | <<additional_information>>
<special-symbol> ::= ';' | ':' | <LF>
<word-symbol> := 'M_Packet' | 'End_M_Packet'
               | 'Profile' | 'End_Profile'
               | 'Identification' | 'End_Identification'
               | 'Identifier' | 'is'
               | 'Module_Algorithm' | 'End_Module_Algorithm'
               | 'imperative' | ''null'' | ''call'' | ''goto''
               | ''abstract'' | 'begin' | 'end' | 'parallel' | 'end_parallel'
               | 'loop' | 'end_loop' | 'condition' | ''while'' | ''until'' | ''for''
               | 'if' | 'then' | 'end_if' | 'exclusive_select' | 'else_if' | 'else'
               | 'end_exclusive_select' | 'case' | 'when'
               | 'inclusive_select' | 'end_inclusive_select'
               | 'terminate' | 'system' | 'module' | 'block'
```

<identifier> ::= '<' <string> '>'
<specification> ::= '[' { <string> | <format_effectors> } ']' [<<additional_information>>]
<description> ::= '[' { <string> | <format_effectors> } ']'
<comment> ::= '--' { <string> | <TAB> } <LF>
<<additional_information>> ::= ''%{'' { <string> | <format_effectors> ''%}''

A <one_byte_code_character> is one of the graphical characters defined in ISO/IEC 646, ISO/IEC 2022, ISO/IEC 4873 and ISO/IEC 8859-1. A <multi_byte_code_character> is another standard character such as ISO/IEC 10646-1. <format_effectors> are control characters defined in ISO/IEC 646. <LF> is LF defined in ISO/IEC 646. <TAB> is HT defined in ISO/IEC 646.

A <word-symbol> is separated from other lexical elements by delimiters ('<', '>', '[',']', '%[', ']%', ';') or by one or more spaces.

<identifier> identifies a part of or a position in a module.

<string> in <specification> describes a specification of a process or a condition.

The description rule of <<additional_information>> and <description> is implementation-dependent.

Delimiters can be inserted into <string> only by putting an escape character '\' immediately in front of them.

5.2 Module Packet

Module Packet is a unit for exchanging tree-structured charts among CASE tools.

<module_packet> ::= 'M_Packet'
<profile_paragraph>
{ <module_identification_paragraph>
<module_algorithm_paragraph> }
'End_M_Packet' ';'

5.3 Profile Paragraph

Profile Paragraph supplements information used to analyze <module_identification_paragraph> and <module_algorithm_paragraph>.

<profile_paragraph> ::= 'Profile'
[<description>]
'End_Profile' ';'

<description> should include:

1. the code set used in <module_identification_paragraph> and <module_algorithm_paragraph>;
2. the maximum string length of <identifier>, <specification>, <<additional_information>>, <description> and <comment>;
3. the type of tree-structured chart; and
4. information about the level of compliance described in clause 4.

5.4 Module Identification Paragraph

Module Identification Paragraph provides the name and the overview of the <module_algorithm_paragragh> following immediately afterwards.

<module_identification_paragraph>
::= 'Identification'
'Identifier' 'is' <identifier> ';'
[<description>]
'End_Identification' ';'

<identifier> shall be unique within Module Packet.

<description> should include:

1. the module specification, such as function, usage, and interface;
2. the module type, such as system, subprogram, task, process, procedure, subroutine, or function; and
3. the programming language for implementing the module.

5.5 Module Algorithm Paragraph

Each Module Algorithm Paragraph describes an algorithm of a tree-structured chart for each module.

<module_algorithm_paragraph> ::= 'Module_Algorithm'
<sequence_of_statements>
'End_Module_Algorithm' ';'

5.5.1 Statement

<sequence_of_statements> ::= { { <<label>> ":" } <statement> }
<statement> ::= <imperative_statement> | <blocked_statement> | <termination_statement>
<<label>> ::= <identifier>

<<label>> identifies an immediately following <statement>. <<label>> is valid only inside the module it is in. <<label>> shall be unique within the <module_algorithm_paragraph>.

5.5.2 Imperative Statement

<imperative_statement> ::= 'imperative' [<<imperative_type>>] <process_specification> ';'
<process_specification> ::= <specification>
<<imperative_type>> ::= <<null>> | <<module_call>> | <<goto>>
<<null>> ::= "null"
<<module_call>> ::= "call" <<module_identifier>>
<<module_identifier>> ::= <identifier>
<<goto>> ::= "goto" <<label>>

<imperative_statement> corresponds to Imperative Construct of ISO/IEC 8631. This statement denotes a process that executes <process_specification> and transfers its control to the next statement.

<<null>> denotes a process that does not execute anything and transfers its control to the next statement.

<<module_call>> denotes a process that transfers its control to the module specified by <<module_identifier>>.

<<goto>> denotes a process that transfers its control to the place specified by <<label>>.

5.5.3 Blocked Statement

<blocked_statement> ::= [<block_identifier>]
["abstract" <<block_specification>>] <compound_statement>
<block_identifier> ::= <identifier>

```
<<block_specification>>  ::=  <specification>
<compound_statement>     ::=  <serial_statement>              |
                              <parallel_statement>            |
                              <iterative_statement>           |
                              <selective_choice_statement>
```

<compound_statement> corresponds to Serial Statement, Parallel Statement, Iterative Statement, or Selective Choice Statement of ISO/IEC 8631.

NOTE <<block_specification>> expresses abstract process descriptions in the stepwise refinement. An example of DXL description with this statement is given in clause A.1.

<block_identifier> identifies the immediately following <compound_statement>. <block_identifier> is valid only inside the module it is in. <block_identifier> shall be unique within the <module_algorithm_paragraph>.

5.5.4 Serial Statement

<serial_statement> ::= 'begin' <sequence_of_statements> 'end' ';'

<serial_statement> corresponds to Serial Construct of ISO/IEC 8631, which denotes a set of sequential processes.

5.5.5 Parallel Statement

<parallel_statement> ::= 'parallel' <sequence_of_statements> 'end_parallel' ';'

<parallel_statement> corresponds to Parallel Construct of ISO/IEC 8631, which denotes a parallel process execution.

5.5.6 Iterative Statement

```
<iterative_statement> ::=
                    <pretested_iteration_statement>    |
                    <posttested_iteration_statement>   |
                    <continuous_iteration_statement>
<pretested_iteration_statement> ::=
                     'condition' [ <<pretested_iteration_type>> ] <condition_specification>
                     'loop'  <sequence_of_statements> 'end_loop' ';'
<posttested_iteration_statement> ::=
                     'loop' <sequence_of_statements>  'end_loop'
                     'condition' [ <<posttested_iteration_type>> ] <condition_specification> ';'
<continuous_iteration_statement> ::= 'loop' <sequence_of_statements> 'end_loop' ';'
<condition_specification> ::= <specification>
<<pretested_iteration_type>>  ::=  ''while'' | ''until'' | ''for''
<<posttested_iteration_type>> ::=  ''while'' | ''until''
```

<iterative_statement> corresponds to Iterative Construct of ISO/IEC 8631. This statement denotes a repeated process with a control condition described by <condition_specification>.

Both <<pretested_iteration_iteration_type>> and <<posttested_iteration_iteration_type>> distinguish control conditions into the following types:

a) while: Continue a loop while the condition is satisfied.
b) until: Stop a loop when the condition is satisfied.
c) for: Continue a loop within the discrete range of <condition_specification>.

5.5.7 Selective Choice Statement

```
<selective_choice_statement> ::=
                <monadic_selective_statement>                |
                <multiple_exclusive_selective_statement>     |
                <multiple_inclusive_selective_statement>
<monadic_selective_statement> ::= 'if' <condition_specification> 'then' <sequence_of_statements>
                                  'end_if' ';'
<multiple_exclusive_selective_statement> ::=
                <if_type_selective_statement>       |
                <case_type_selective_statement>
<if_type_selective_statement> ::=
            'exclusive_select'
                  'if' <condition_specification> 'then' <sequence_of_statements>
                { 'else_if' <condition_specification> 'then' <sequence_of_statements> }
                [ 'else' <sequence_of_statements> ]
            'end_exclusive_select' ';'
<case_type_selective_statement> ::=
            'exclusive_select' 'case' <expression_specification>
                  { 'when' <choice_specification> ':' <sequence_of_statements> }
            'end_exclusive_select' ';'
<expression_specification> ::= <specification>
<choice_specification>     ::= <specification>
<multiple_inclusive_selective_statement> ::=
            'inclusive_select' 'case' <expression_specification>
                  { 'when' <choice_specification> ':' <sequence_of_statements> }
            'end_inclusive_select' ';'
```


<selective_choice_statement> corresponds to Selective Choice Construct of ISO/IEC 8631, which denotes a conditional selection of processes.

<multiple_exclusive_selective_statement> corresponds to Dyadic Selective Construct and Multiple Exclusive Selective Construct of ISO/IEC 8631.

DXL provides two types of statements to describe a condition of the multiple exclusive selective choice: <if_type_selective_statement> and <case_type_ selective_statement>.

<if_type_selective_statement> selects the <sequence_of_statements> that corresponds to the first <condition_specification> having a boolean value ''true'' in the enumerated conditions. If every <condition_specification> has a boolean value ''false'', the <sequence_of_statements> immediately following 'else' is selected.

<case_type_selective_statement> selects the <sequence_of_statements> that corresponds to the first <choice_specification> coinciding with <expression_specification> in the enumerated conditions.

<multiple_inclusive_selective_statement> corresponds to Multiple Inclusive Selective Construct of ISO/IEC 8631.

5.5.8 Termination Statement

```
<termination_statement> ::= 'terminate' <termination_target> <specification> ';'
<termination_target> ::= 'system' | 'module' | 'block' <block_identifier>
```

<termination_statement> corresponds to Termination of ISO/IEC 8631.

If <termination_target> is 'system', all modules related to this termination stop here.

If <termination_target> is 'module', the module that contains this termination stops here and returns its control to the calling module.

If <termination_target> is 'block', the block identified by <block_identifier> stops here and transfers its control to the next statement of this block.