TECHNICAL REPORT

ISO TR 10562

First edition
1995-05-15

Manipulating industrial robots — Intermediate Code for Robots (ICR)

Robots manipulateurs industriels — Codification intermédiaire pour robots (ICR)

Reference number
ISO/TR 10562:1995(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The main task of technical committees is to prepare International Standards. In exceptional circumstances a technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/TR 10562, which is a Technical Report of type 2, was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 2, *Robots for manufacturing environment*.

This document is being issued in the type 2 Technical Report series of publications (according to subclause G.4.2.2 of part 1 of the ISO/IEC Directives, 1992) as a "prospective standard for provisional application" in the field of manipulating industrial robots because there is an urgent need for guidance on how standards in this field should be used to meet an identified need.

This document is not to be regarded as an "International Standard". It is proposed for provisional application so that information and experience of its use in practice may be gathered. Comments on the content of this document should be sent to the ISO Central Secretariat.

A review of this type 2 Technical Report will be carried out not later than two years after its publication with the options of: extension for another two years; conversion into an International Standard; or withdrawal.

Annexes A to D of this Technical Report are for information only.

This page intentionally left blank

Manipulating industrial robots — Intermediate Code for Robots (ICR)

1 Scope

This Technical Report specifies an intermediate code to be used as a neutral interface between user-oriented robot programming systems and industrial robot control systems and to define the structure of and the access mechanisms to data lists, which contain data accompanying the intermediate code.

2 Normative references

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

ISO 8373 : 1994, *Manipulating industrial robots - Vocabulary.*

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

ISO/IEC 9506-3 : 1991, *Industrial automation systems - Manufacturing message specification - Part 3: Companion standard for robotics.*

3 Definitions

For the purposes of this Technical Report, the definitions given in ISO 8373, and the following, apply.

All code and data elements will be represented in accordance with ISO 8859-1 (8-bit character set). ICR is machine or interpreter-oriented instead of user-oriented.

The following words are used in the specialized definition in this Technical Report.

3.1 ICR program: Collection of robot directives and data written in ICR.

3.2 data list: Collection of positions, poses and other data structures written in ICR codes.

3.3 robot control unit: Controlling device which accepts instruction codes and sends servo signals to each joint of an industrial manipulator.

4 Compliance with this Technical Report

A program code complies with this Technical Report if it uses only the features of the code that are defined by this Technical Report and it does not rely on any particular interpretation of implementation-dependent features.

The single statements of the Technical Report can be divided into two groups.

The first group (A) contains the statements whose implementation is independent of the industrial robot control and the robot type to be controlled.

The second group (B) contains the statements that are dependent on a specific industrial robot type and robot control.

The group A is divided into 8 compliance levels.

The basic level (level 0) of group A contains:

```
 Memory and data management
 Program flow control
 Boolean and arithmetic operations
 The statements
    CNFGCHAN, OPENCHAN, CLOSECHAN, RESETCHAN, STATCHAN,
    DIN and DOUT  of the exchange of datastatements
 Each addressing mode except symbolic addressing
```

As extension of the basic level there are 7 compliance levels, which can extend the basic level independent from each other.

1	Symbolic addressing mode
2	Exchange of data statements except CNFGCHAN, OPENCHAN, CLOSECHAN, RESETCHAN, STATCHAN, DIN and DOUT
3	Data list management
4	Sensor functions
5	Debugging functions
6	User specific extensions
7	Future extensions

The group B is divided into a kernel and an extension.

The kernel of group B contains:

```
ACCEL, VEL, and MOVTIM statements
JMOVE and LMOVE statements
CALIB, CURPOS, GOHOME, MOVSTP, MOVCONT, and MOVCANCEL
   statements
Description of end effector facilities
```

The extension of group B contains:

```
Each technical specification not contained in the kernel
Each move and effector control not contained in the kernel
Description of robot facilities
```

An ICR processor is defined by this Technical Report to be "a system or mechanism that accepts an ICR code as input and executes ICR code according to the semantics defined by the standard". A processor complies with the standard if it satisfies at least the basic level of group A and the kernel of group B, except of those functions of the kernel of group B that are explicitly reported by the processor to be not accepted.

ICR may be represented in two different forms.

In the first form all operators only consist of ASCII-character digits. The second form allows mnemonics to express the operators. The effect of the following two statements is exactly the same:

a) 220, 5450;

and

b) 220, 'MOVEND';

In statement a) the operator is a number, while in b) the operator is in the form of a mnemonic.

Due to this dual representation of ICR-code there are two different kinds of ICR-interpreters. The first and simpler kind needs numerically represented operators, while the second kind also allows using mnemonics. This second kind, called MNEMONICAL-ICR-interpreter, may also be interpreted as a kind of preprocessor to convert mnemonics into numerical values which are reprocessed by a simple ICR-interpreter of the first form.

5 Basic concepts

5.1. Introduction

ISO/TR 10562 is part of a series of international standards dealing with the requirements of manipulating industrial robots. Other documents cover such topics as safety, general characteristics, performance criteria and related testing methods, terminology, and mechanical interfaces. It is noted that these standards are interrelated and also related to other International Standards.

This Technical Report describes a second committee draft for an intermediate code ICR between off-line programming and different robot control units. There are various levels for the representation of robot programming languages. ICR allows the interchange of technological and geometrical data between robot control units also. This does not mean, that other levels of intermediate code are inhibited. This proposal doesn't eliminate another intermediate code between ICR and robot control units. Nevertheless, ICR should be used as a standardized neutral interface between robot programming and robot control units.

Annex A of this Technical Report provides guidance to the robot system state variables, defined in ISO/IEC 9506-3 "Industrial automation systems - Manufacturing specification - Part 3: Companion standard for robotics", which are referred to in ICR.

Annex B of this Technical Report provides information on how to use and implement ICR at the application and how to come from a robot programming language to ICR code.

A manipulating industrial robot has to perform various tasks. The set of motions and auxiliary function instructions which define a specific intended task of the robot system is specified by a task program.

A given application is specified in terms of task description, or in terms of program, which is written in a user-oriented programming language. The syntax and semantic of this user-oriented high-level programming language depends on the type of system programming, e.g. declarative programming, CAD-oriented interactive programming. The program has to be translated into the low level program code of the robot and then loaded into the robot control unit which will execute it.

According to the number of various robot controllers, and to the evolution of user-oriented high-level programming systems, it is necessary to build a bridge between both kinds of languages.

Such a gateway is called **ICR** (**I**ntermediate **C**ode for **R**obots). Its goal is to support all the features that a robot control usually performs and all the elements to describe a task.

The ICR code is useable to exchange user programs and data between different robot control units and/or between any robot programming system and control. The code itself consists of a sequence of records representing different instructions. Such an instruction may be a data or type definition also. The instructions include arithmetic and stack operations, which allow an efficient calculation of arithmetic expressions with respect to a postfix notation.

The block structure of a high-level programming language is supported which helps for structured programming. Variables or other programming objects are identified by symbols with respect to their scope in the block structure. For some special use, absolute addresses may be used, e.g. to handle memory mapped I/O directly.

The proposal shows the scope of work and the functions of ICR in detail, but the formal description is performed on syntax resp. mnemotechnical level. The definition of code numbers is an easy mapping to the ICR elements described in this paper.

Some of the used items like pose or orientation are not explained in detail. Their meaning is described in other ISO documents.

The main features of the intermediate code are as follows:

- Intermediate codes: The position of the code is the standard output format from the robot programming systems and the standard input format to the robot control units as illustrated in figure 1.

```
                      |user oriented
                      |high level
                      |robot
                      |programming
                      |language
                      |
                      v
               -------------
              |    robot     |
              | programming  |
              |    system    |
               -------------
                   |     ^
                   |     |no way return
                   |ICR  |(except data lists)
                   |     |
                   |     |             |ICR
                   v     |             v
 ---------     ------------        ------------
| teach   |-->|   robot    |<--->|   robot    |
|   in    |   |  control   | data|  control   |
| module  |   |   unit 1   | list|   unit 2   |
 ---------     ------------   ex-  ------------
                    ^       change
                    |
                    v
            ------------------
           |    industrial    |
           |       robot      |
            ------------------
```

Figure 1 - Structure of robot programming with neutral interfaces

- The standard format for data exchange: Data list written in ICR can be used for data exchange and for input data of teaching-by-showing.

- Machine-machine interface: ICR is designed as data exchange format between computer systems. As it is not a user-oriented language, the codes can be represented basically by integer only. However, it uses symbols for variables and others to improve the readability of the program. Printable character set is used in ICR.

Both data and program written in ICR are transferred among the systems in figure 1. The Program is executed in the following two ways:

- a program written in ICR is once converted into that in a user-specified code by means of a translator, and the converted program is transferred to the robot control unit.
- the program written in ICR is transferred to the robot control unit in file or through a communication line, and is directly executed on the robot control unit.

The system configuration and the implementation of a robot control unit is fully dependent on the system developer of the robot. In this Technical Report, however, the robot control unit is assumed to have the structure indicated in figure 2.

- path generator: it computes the trajectory of the industrial manipulator.
- servo system: it realizes the path through servo system.
- peripheral controller: it controls digital/analog input/output units.
- sensors: no specified devices are defined. Instead, general DIO is assumed.
- communication line: a serial line such as RS232C is assumed.
- file storage system: files can be used as option.

```
robot control unit
     |
     --- path generator
     |       |
     |        --- servo system --- joints
     |
     --- peripheral controller
     |
     --- sensors
     |
     --- communication line
     |
     --- file storage system
```

Figure 2 - Configuration of a robot control unit

As for an industrial manipulator, an articulated robot with 6 degrees of freedom is assumed. Any robot with redundant degrees of freedom is not covered by this Technical Report. When a robot with less than 6 degrees of freedom is used, it is defined in the same manner as well.

5.2 Description of ICR grammar

At the lexical level, an ICR program consists of a sequence of characters of the ISO 8859-1 (8-bit character set) norm. All control characters (such as a linefeed or a carriage return) are ignored in processing.

5.2.1 Metalanguage of ICR

The metalanguage for the lexical definition of ICR is derived from the BNF notation:

- terminal symbols (here characters) are enclosed in quotation marks (');
- parentheses '(' and ')' are used to show grouping;
- concatenation is indicated by the juxtaposition of (terminal or non-terminal) symbols;
- optional symbols are indicated by the square brackets '[' and ']';
- repetition is indicated by the braces '{' and '}';
  {item1} means: *item1* occurs not, one time, two times, ... , or n-times
- alternation is indicated by the stroke '|'
  item1 | *item2* means: *item1* or *item2* but not both

Example:

```
number_constant ::= '#' (integer | real) .
```

A number constant can be #30 to define the integer 30 or #0.3E2 to define the real number 30.0.

5.2.2 ICR lexical elements

The syntax in this clause describes the formation of lexical elements out of characters and the separation of these elements from each other. Therefore, the syntax is not of the same kind as that used in the rest of this standard. That means, the ICR lexical elements differ from the syntactical definition because no space character is allowed between the terminal symbols of a repetition, e.g. the definition of an integer. The lexical elements will be referenced by the syntactical definitions in the following chapters.

```
digit ::=
     '0' | '1' | '2' | '3' | '4' |
     '5' | '6' | '7' | '8' | '9' .

upper_case_letter ::=
     'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' |
     'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' |
     'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' |
     'V' | 'W' | 'X' | 'Y' | 'Z' .

lower_case_letter ::=
     'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' |
     'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' |
     'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' |
     'v' | 'w' | 'x' | 'y' | 'z' .

letter ::= upper_case_letter | lower_case_letter .
```

```
space ::=  ' ' .

underscore ::=  '_' .

special_character0 ::=
     '!' | ' " ' | '#' | '$' | '%' | '&' | ''' |
     '(' | ')' | '*' | '+' | ',' | '-' | '.' | '/' .

special_character1 ::=
     ':' | ';' | '<' | '=' | '>' | '?' | '@' .

special_character2 ::=
     '[' | ']' | '\' | '^' | '`' .

special_character3 ::=
     '{' | '|' | '}' | '~' .

graphic_character ::=
     letter | digit | space | underscore |
     special_character0 | special_character1 |
     special_character2 | special_character3 .

unsigned_integer ::=  digit { digit } .
```

The semantic of the data types integer, character, real, boolean, and string is described in subclause 6.2.

```
integer ::=  [ ( '+' | '-' ) ] unsigned_integer .

character ::=
     ''' graphic_character ''' |
     '$' unsigned_integer .
```

In the description of character, dollar mark '$' is used to describe the character code number. The type of the character code number is the ISO 8-bits code 8859-1.

```
real ::= integer '.' [unsigned_integer] ['E' integer] .

boolean ::= 'T' | 'TRUE' | 'F' | 'FALSE' .
```

A **string** is a series of graphic characters surrounded by double quotation marks ' " '. To include a double quotation mark itself in a string, a concatenation of double quotation marks is used such as "abc""def" standing for abc"def.

```
string ::= ' " ' {graphic_character} ' " ' .
```

A **symbol** is formed similar to a string with no restriction on its length in which all characters are either letters, digits or underscore and the first character is a letter. There is no difference between small and capital letters (i.e. ABC and abc refer the same object). A symbol is not surrounded by double quotation marks.

```
symbol ::= letter { ( letter | digit | underscore ) } .
```

5.3 Address modes

5.3.1 Absolute addresses

When users need to specify absolute addresses of robot control units and other peripherals, they can use integer numbers which means absolute addresses of the devices. The addresses should be accepted by the robot control unit. Note that the interpretation of the absolute address is completely system dependent. Therefore users are strongly requested not to use the absolute addresses in order to make the codes portable.

As mentioned already, the computational environment of a robot control unit is out of the scope of this standard. However, we often need to have the image of the memory space, e.g. I/O mapped peripherals, because ICR is a simple, and rather primitive language. Note that the "absolute addresses" are not in every case real addresses, but virtual addresses of the virtual memory space of the CPU. Its data type is positive INTEGER with no restrictions except the range of INTEGER type. It can be denoted as

```
absolute_address ::= '@' unsigned_integer .
```

where unsigned_integer represents the address of virtual memory space.

Due to very high robot control system dependency the usage of absolute addresses should be avoided.

5.3.2 Block relative addresses

A block relative address refers to a variable which is declared for a program block and its inner blocks only. If the block relative address is given by an integer, its value is calculated by the definition formula

```
            2^24*block_nesting_number + variable_index
```

(Remark: The number 2^{24} was defined with respect to a representation of a block relative address by an integer of 32 bits). The block_nesting_number refers to an identical number of block nesting given in the block begin statement (BLKBEG) of a block activated before. If several blocks with the same number of block nesting are activated simultaneously, the block that was activated at last is assumed. The block nesting number starts with zero for the main program.

The variable_index refers to an identical block relative index of a variable declared by a declaration of variable statement (DECLVAR) inside the block referred to by the block_nesting_number.

If a block relative address is given by its symbol, the outer block or procedure name is noted as a prefix with a point, e.g. procedure_out.variable_name. If no block or procedure name is written, the inner block is assumed.

```
blockrel_address ::=
      'BRADR' '(' (integer | [ symbol '.' ] symbol) ')' .
```

5.3.3 Access to operands

In ICR instructions, the access to operands is performed by using constants, absolute or block relative addresses, or symbols.

```
operand ::=
      symbol | absolute_address | constant | blockrel_address .
```

Part of the operands are pushed on the stack and the rest of them are described in the statements. The results of the execution of instructions are pushed on the stack, if any. When a symbol appears, it is evaluated as a label, a procedure name, or a variable, according to the definition of the instruction.

Constants of integer type are denoted by "#1012", where '#' is ASCII character. "#1012" is the decimal number of one thousand and twelve.

To indicate the data pushed on and popped from the stack, the following notation is used in the definition of each instruction:

```
Stack:
      I_n: var_type, .... , I_2:var_type, I_1:var_type
=>    O_m: var_type, .... , O_2:var_type, O_1:var_type
```

where I_x indicates an x-th input value and O_x indicates an x-th output value, and var_type means the type of data.

The stack is "Last In First Out (LIFO)" memory and the orders of input/output values are illustrated in figure 3. Therefore, I_1 and O_1 is located on the top of the stack (TOS).

```
           ---------
  TOS -->|   I_1   |
           ---------
         |   I_2   |                  ---------
           ---------        TOS -->|   O_1   |
              .                       ---------
              .        =====>      |   O_2   |
              .                       ---------
         |         |                     ...
           ---------                  ---------
         |   I_n   |               |   O_m   |
      -----------------         -----------------
```

Figure 3 - Execution of stack operation

"Var_type" can be one or more data type names (see "Variables and constants"). When two or more data types are accepted, express them with a pair of parentheses and a vertical stroke as shown in the following way:

```
I_1:(var_type1 | var_type2) .
```

This notation is out of the lexical definition of ICR, and is only used for the explanation and description of the functionality.

5.4 Structure of ICR units

The basic structure of an ICR program consists of units. A unit may include a user program, that can also be a module containing procedures and functions, or a data list including data. Modules are needed to develop program parts separately and link them together. Data lists can be used to exchange data from one robot control unit to another, e.g. the positions for a user program.

```
icr_unit ::= program | data_list .

program ::= pbeg { statement } pend .

data_list ::= dlhead { dldat } dlend .
```

5.5 Statement structure

Every operation expressed by ICR is written as a statement consisting of a line number and an instruction. An instruction consists of an operator and zero, one or more operand(s). The operands are separated by a comma and the operation is closed by a semicolon.