Wulf Werum/Hans Windauer

Introduction to

PEARL

Process and Experiment Automation Realtime Language

Programm
Angewandte Informatik

Vieweg

Wulf Werum
Hans Windauer

Introduction to PEARL

Process and Experiment Automation Realtime Language

Description with Examples

Friedr. Vieweg & Sohn Braunschweig/Wiesbaden

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Werum, Wulf:
Introduction to PEARL: process and experiment automation realtime language; description with examples/Wulf Werum; Hans Windauer. –
Braunschweig; Wiesbaden: Vieweg, 1982.
(Programm Angewandte Informatik)
ISBN 3-528-03590-0

NE: Windauer, Hans:

Printed by W. Langelüddecke, Braunschweig
Bookbinder: W. Langelüddecke, Braunschweig
Cover design: Peter Morys, Wolfenbüttel
Printed in Germany

ISBN 3-528-03590-0

PREFACE

PEARL (Process and Experiment Automation Realtime Language)
is a general purpose high order language designed to meet the requirements of realtime programming in all fields of process and experiment automation by means of computers.

PEARL has been developed in the period from 1969 until 1976 by cooperation of German manufacturers (e.g. AEG , BBC , Siemens), software and systems houses (e.g. ESG , GEI , GPP , IDAS , mbp , Werum), scientific institutes (e.g. Hahn-Meitner-Institute of Berlin, Nuclear Research Centre of Jülich, Universities of Erlangen, Heidelberg and Stuttgart) and users, organized by GMR - VDI/VDE, the German association of engineers. The development has been managed by the project PDV of the Kernforschungszentrum Karlsruhe GmbH; it has been supported by the Federal Ministry of Research and Technology of Germany. More detailed information about history and applications of PEARL can be read in Martin 78 and Martin 81.

In mid 1978 the draft standard of Basic PEARL was published (Standard 78). This standard defines that subset of the full language PEARL, which has to be common to all PEARL implementations. The draft standard of Full PEARL was finished in mid 1980 (Standard 80).

Since 1979 the PEARL Association cares about PEARL activities of common interest, especially about interests of PEARL users.

This language reference manual describes a subset of Full PEARL, which contains in addition to Basic PEARL those language features of Full PEARL necessary for elegant and efficient programming of complex applications. In principal only those features of Full PEARL are not considered, which require additional storage and administration effort at runtime. The extensions and restrictions in contrast to Basic PEARL and Full PEARL, resp., are listed in the appendix.

For this language Werum developed a portable compiler, which has been used by Werum and other institutions to implement the language on the computers

- Amdahl 470/6 (for cross compilation)
- Hewlett Packard HP 1000
- Hewlett Packard HP 3000
- INTEL 8086
- Norsk Data NORD 10 , NORD 100
- RDC (a Really Distributed Computer Control System; see Syrbe 78 and Steusloff 80)
- Siemens 404/3
- Siemens 310
- Siemens 330, R 30
- Siemens 7.760 (for cross compilation).

The manual consists mainly of three parts: first, the introduction characterizes the "new" features of PEARL, i.e. the most important differences to older high order languages like FORTRAN or PL/I. For reasons of good understanding the second part describes only those features necessary to write simple programs. Additional possibilities are presented in the third part, which is followed by selected references to literature. For further references see Martin 78, Martin 81 and Kappatsch 79.

The appendix contains a list of all keywords, a table showing the valid possibilities to use the data types and a complete syntax description. (In order not to confuse the reader, the paragraphs often don't define the complete syntax of a language feature.) Paragraph 4 and 5 of the appendix list the extensions and restrictions in contrast to Basic PEARL and Full PEARL , respectively. The manual is closed by an index.

The language reference manual presented here is a translation of Werum 78 taking into account most recent standardization results (Standard 80).

CONTENTS

Part I

Introduction

1. IMPORTANT LANGUAGE FEATURES

1.1 Realtime Features

A program for on-line control or evaluation of a technical process has to react rapidly on spontaneously received information of the process or on timely results. Out of this reason it is not sufficient to arrange and go through the various divisions of the program sequentially, that means in timely unchanged sequence. It is of importance that the more or less complex automation problem has to be divided into problem-justified components of different states of urgency and that the program structure must be fitted to this problem structure. This causes the existence of independent program elements for sub-problems ready to be solved timely sequentially among other problems (e.g. procedures). However, there also arise independent program elements for sub-problems, which based on a timely not determined cause (e.g. disturbance in the process under control) have to be solved immediately parallel to other problems.
The execution of such a program element is called "task", for determination of urgency such tasks can be provided with priorities.

Concerning the definition and combination of tasks - with regard to the technical process - PEARL offers the following possibilities:

- Definition of tasks, e.g.

```
SUPPLY: TASK PRIORITY 2;
        taskbody (definitions, statements)
        END;
```

- Start (activation), e.g.

```
ACTIVATE SUPPLY;
```

- Termination, e.g.

```
TERMINATE PRINTING;
```

- Suspension, e.g.

```
SUSPEND STATISTICS;
```

- Continuation, e.g.

  CONTINUE STATISTICS;

- Delay, e.g.

  AFTER 5 **SEC RESUME;**

According to demands of automation problems some of these statements can be scheduled for their (repeated) execution, e.g. scheduled for the case of time entrance, the end of a duration or the occurrence of an interrupt:

WHEN READY **ACTIVATE** SUPPLY;

(Meaning: Each time when the interrupt READY occurs, the task SUPPLY has to be activated.)

Schedules can also determine the timely periodical start:

AT 12:0:0 **EVERY** 30 **MIN UNTIL** 15:0:0 ACTIVATE PROTOCOL;

As far as certain actions do not interfere, different tasks execute their statements independently of each other. Sometimes however synchronization of two or more tasks is required, e.g. if a task periodically creates data for other tasks and puts them into a buffer. In this case the producer is not allowed to work faster than the consumer.

Synchronization problems of higher complexity arise, if a task must have exclusive access to a file (when writing), while others also participate simultaneously (when reading).

In order to solve such synchronization problems PEARL contains the synchronization primitives sema and bolt variables.

1.2 Input and Output

In order to cope with devices of standard periphery (printer, card reader, disc etc.) or process periphery (measurement points, valves and so on) as well as with the administration of files PEARL provides computer independent statements.

Devices and files are being summarized with the term data station.
In general there exist two kinds of data transfer:

- The transfer of data without transforming them to (or from) external representation:
  This kind of data transfer is provided for file handling allowing sequential and direct access and for transfer of process data.

  Examples:

  ```
  READ ARTICLE FROM ARTICLEFILE BY POS(10);
  SEND OFF TO MOTOR(1);
  ```

- The transfer of data with transforming them to (or from) external representation:
  This means for example the representation of data with characters of the character set of the data station.

  Example:

  ```
  PUT RESULT TO PRINTER BY F(5);
  ```

The names of data stations are free-to-be-chosen. This results by dividing a PEARL program into computer dependent and mostly computer independent divisions.

1.3 Program Structure

Program systems for solving highly complex automation problems should be modular. PEARL meets this requirement based on the fact that a PEARL program is composed of one or several independently compilable units, the so-called modules. In order that statements for input/output as well as for handling events in the technical process (interrupts) or in the computer system (signals) can be programmed computer independently, generally a module consists of a system division and a problem division.

The hardware configuration is described in the system division. In particular, freely-chosen user identifications may be attached to devices, interrupts, and signals.

In the following example a valve is connected with the connection point 3 of a digital output device which has the computer-dependent "system identification" DIGOUT(1). The valve, i.e. the connection point 3 of DIGOUT(1), shall become the freely-chosen "user identification" VALVE.

```
VALVE: DIGOUT(1) * 3;
```

Now it is possible to program the algorithm for solving the input/output problem computer independently in the problem division by using the user identification introduced in the system division, e.g.:

```
TAKE STATUS FROM VALVE;
```

2. RULES FOR THE CONSTRUCTION OF PEARL PHRASES

A PEARL program can be written without usage of special program-forms; no special attention has to be paid to the fact that a statement begins on a certain line.

All elements of a PEARL program are composed of the following characters. Character string denotations and comments are permitted to be composed of each character, which is accepted by the machine configuration.

2.1 Character Set

The character set of PEARL contains the following elements:

- capital letters A - Z,
- the digits 0 to 9 and
- the special characters
  - ␣ blank, space,
  - ' apostrophe,
  - (left parenthesis,
  -) right parenthesis,
  - , comma,
  - . period, pot,
  - ; semicolon,
  - : colon,
  - + plus sign,
  - - minus sign, hyphen,
  - * asterisk,
  - / slash,
  - = equal-sign,
  - < left angle bracket,
  - > right angle bracket,
  - [left bracket,
  -] right bracket

The following combinations of special characters are being interpreted as one unit:

** exponentiation operator
/* begin of a comment,
*/ end of a comment,
// integer division operator,
== operator equal,
/= operator not equal,
< = operator less or equal,
> = operator greater or equal,
< > operator cyclic shift,
>< operator concatenation,
:= assignment symbol,
<- transfer direction symbol: arrow left,
<-> transfer direction symbol: double arrow,
-> transfer direction symbol: arrow right.

In case there aren't all of these symbols available on the device concerning program-writing, the following characters could be used alternatively:

LT	for <
GT	for >
NE	for /=
LE	for < =
GE	for > =
CSHIFT	for <>
CAT	for ><
(/	for [
/)	for]