TECHNICAL REPORT

IEC TR 61499-3

First edition
2004-06

Function blocks for industrial-process measurement and control systems –

Part 3: Tutorial information

Reference number
IEC/TR 61499-3:2004(E)

Publication numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series. For example, IEC 34-1 is now referred to as IEC 60034-1.

Consolidated editions

The IEC is now publishing consolidated versions of its publications. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Further information on IEC publications

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology. Information relating to this publication, including its validity, is available in the IEC Catalogue of publications (see below) in addition to new editions, amendments and corrigenda. Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is also available from the following:

- **IEC Web Site (www.iec.ch)**

- **Catalogue of IEC publications**

  The on-line catalogue on the IEC web site (www.iec.ch/searchpub) enables you to search by a variety of criteria including text searches, technical committees and date of publication. On-line information is also available on recently issued publications, withdrawn and replaced publications, as well as corrigenda.

- **IEC Just Published**

  This summary of recently issued publications (www.iec.ch/online_news/ justpub) is also available by email. Please contact the Customer Service Centre (see below) for further information.

- **Customer Service Centre**

  If you have any questions regarding this publication or need further assistance, please contact the Customer Service Centre:

  Email: custserv@iec.ch
  Tel: +41 22 919 02 11
  Fax: +41 22 919 03 00

TECHNICAL REPORT

IEC TR 61499-3

First edition
2004-06

Function blocks for industrial-process measurement and control systems –

Part 3: Tutorial information

International Electrotechnical Commission, 3, rue de Varembé, PO Box 131, CH-1211 Geneva 20, Switzerland
Telephone: +41 22 919 02 11 Telefax: +41 22 919 03 00 E-mail: inmail@iec.ch Web: www.iec.ch

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

PRICE CODE X

For price, see current catalogue

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

FUNCTION BLOCKS FOR INDUSTRIAL-PROCESS MEASUREMENT AND CONTROL SYSTEMS –

Part 3: Tutorial information

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

The main task of IEC technical committees is to prepare International Standards. However, a technical committee may propose the publication of a technical report when it has collected data of a different kind from that which is normally published as an International Standard, for example "state of the art".

IEC 61499-3, which is a technical report, has been prepared by working group 6: Function blocks, of IEC technical committee 65: Industrial-process measurement and control systems.

The text of this technical report is based on the following documents:

Enquiry draft	Report on voting
65/308/DTR	65/321/RVC

Full information on the voting for the approval of this technical report can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

IEC 61499 consists of the following parts under the general title *Function blocks for industrial-process measurement and control systems:*

Part 1: Architecture

Part 2: Software tools requirements

Part 3: Tutorial information

Part 4: Communication requirements

NOTE Parts 1 and 2 are under consideration.

The committee has decided that the contents of this publication will remain unchanged until the maintenance result date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed;
- withdrawn;
- replaced by a revised edition, or
- amended.

A bilingual version of this publication may be issued at a later date.

INTRODUCTION

The following gives a description of the contents of the various parts of IEC 61499.

a) Part 1 contains

- general requirements, including an introduction, scope, normative references, definitions, and reference models;
- rules for the declaration of function block types, and rules for the behaviour of instances of the types so declared;
- rules for the use of function blocks in the configuration of distributed industrial-process measurement and control systems (IPMCSs);
- rules for the use of function blocks in meeting the communication requirements of distributed IPMCSs;
- rules for the use of function blocks in the management of applications, resources and devices in distributed IPMCSs.

b) Part 2 defines requirements for software tools to support the following systems engineering tasks enumerated in 1.1 of IEC 61499-1:

- the specification of function block types;
- the functional specification of resource types and device types;
- the specification, analysis, and validation of distributed IPMCSs;
- the configuration, implementation, operation, and maintenance of distributed IPMCSs;
- the exchange of information among software tools.

c) Part 3 has the purpose of increasing the understanding, acceptance, and both generic and domain-specific applicability of IPMCS architectures and software tools meeting the requirements of the other parts, by providing

- answers to Frequently Asked Questions (FAQs) regarding IEC 61499;
- examples of the use of IEC 61499 constructs to solve frequently encountered problems in control and automation engineering.

d) Part 4 defines rules for the development of compliance profiles which specify the features of IEC 61499-1 and IEC 61499-2 to be implemented in order to promote the following attributes of IEC 61499-based systems, devices and software tools:

- interoperability of devices from multiple suppliers;
- portability of software between software tools of multiple suppliers; and
- configurability of devices from multiple vendors by software tools of multiple suppliers.

FUNCTION BLOCKS FOR INDUSTRIAL-PROCESS MEASUREMENT AND CONTROL SYSTEMS –

Part 3: Tutorial information

1 Scope

This part of IEC 61499, which is a technical report, is intended to provide a simple shorthand for common functionality in a broad number of "application domains" and, to that extent, may be considered a "language". It should be noted that IEC 61499 is not a programming methodology *per se*.

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 61131-3:2003, *Programmable controllers – Part 3: Programming languages*

IEC 61804-1:2003, *Function blocks (FB) for process control – Part 1: Overview of system aspects*

IEC 61804-2:2004, *Function blocks (FB) for process control – Part 2: Specification of FB concept and Electronic Device Description Language (EDDL)*

3 Frequently asked questions (FAQ)

3.1 General questions

What is this clause about?

This clause is a compilation of responses to FAQ about the various parts of IEC 61499.

What good is IEC 61499?

IEC 61499-compliant distributed industrial-process measurement and control systems (IPMCSs), devices, and their associated life-cycle support systems will be able to deliver a number of significant benefits to their owners and system integrators, including:

a) IEC 61499-compliant life-cycle support systems will be able to **reduce engineering costs** through integrated facilities for configuration, programming and data management. Additional engineering cost savings will result from the **ease of system integration** provided by IEC 61499's simple yet complete model of distributed systems. This model provides hardware- and operating-system-independent representations for all functions of the system, including control and information processing as well as communications and process interfaces.

b) Engineers and technicians will be able to **reduce system implementation time** by applying a common set of concepts and skills to all elements of the system. Further reductions in implementation time will result from the elimination of software patches and "glueware" formerly required to integrate incompatible system elements and software tools.

   1) The elimination of patchware and glueware, the availability of interoperable software tool sets, the portability of engineering skills, and the ease of integration of system elements, will yield **higher reliability and maintainability** over the system life cycle.

2) IEC 61499 provides an abstract, implementation-independent means of representing system functions. This common "target" will lead to **simplified migration from existing systems** to IEC 61499-compliant systems, and from older to **newer technological platforms** (operating systems, communications, etc.) in IEC 61499-compliant systems.

3) Economies of scale from uniform application of common software and firmware technologies will provide **lower hardware cost per function**, since the most significant cost items in modern control hardware are its firmware and supporting software.

Is IEC 61499 a "stand-alone" document?

No. In order to realize the benefits listed above, distributed industrial-process measurement and control systems (IPMCSs) will need

- **compliance profiles** following the rules given in IEC 61499-4, including full definitions of the communication protocols utilized;
- **standard programming languages** such as those defined in IEC 61131-3 for the specification of algorithms in basic function block types;
- **libraries** of standardized and customized function block types, resource types and device types and guidelines for their application in specific domains.

How can IEC 61499 function blocks be distinguished from IEC 61131-3 function blocks?

Unfortunately, the term was already used differently in IEC 61131-3 and in the process control domain as reflected in IEC 61804. IEC 61499 defines the term most generically in terms of a distributed, event-driven architecture, of which the centralized, scanned architecture underlying IEC 61131-3 and the distributed, scanned architecture underlying IEC 61804 may be considered special cases. IEC 61131-3 and IEC 61804 may, in future, choose to specify their own **compliance profiles** following the rules of IEC 61499-4 in order to provide full harmonization of these standards.

3.2 Object orientation

Why use such a heavily object-oriented model?

The degree of object orientation used in IEC 61499 is necessary in order to achieve the required level of distributability of encapsulated, reusable software modules (function blocks).

The benefits of using object-oriented development, and the extent to which IEC 61499 realizes these benefits, are discussed in Annex B of this document.

Is this not very expensive to implement?

Not necessarily; a full object-oriented implementation is not required by the IEC 61499 model – only that the externally visible behaviour of function blocks and compliant devices conform to the requirements of IEC 61499 and possibly of compliance profiles as defined in IEC 61499-4.

Why not use a general-purpose distributed object model, like DCOM or CORBA?

Implementation of the features specified for these information-technology models would be too expensive, and their performance would almost always be too slow for use in a distributed real-time industrial-process measurement and control system (IPMCS).

Additionally, there is no standard, easily understood graphical model for representing the interconnections of events and data among these kinds of objects in distributed applications.

Are data connections and event connections a kind of object?

The **declarations** of data and event connections contained in **library elements** can be considered objects to be manipulated by software tools, as shown in Annex C of IEC 61499-1. Additionally, data and event connections are regarded as **managed objects** within **resources**, as shown in Annex C of IEC 61499-1.

Why are there no GLOBAL or EXTERNAL variables?

All variables are encapsulated. There is no guarantee that there will be an implicit global distribution mechanism available. When such mechanisms are available, they can always be mapped to service interface function blocks.

How can "contained parameters" be accessed?

External access to internal variables of function blocks is contrary to the principles of good software design. Nonetheless, to accommodate exceptional cases and previous practice, the READ and WRITE services and associated access paths to internal variables are defined for management function blocks. However, this practice may substantially degrade system performance, reliability, maintainability and safety, especially if used instead of the standard PUBLISH/SUBSCRIBE or CLIENT/SERVER services for high-rate, periodic access to variables.

Why use function blocks to model device or resource management applications?

The benefits of this approach are

- a consistent model of all applications in the system, including management applications;
- a consistent means of encapsulating and reusing all functions, including management functions;
- reuse of existing data types;
- use of existing, standardized means for the definition of required new data types and specification of management messages.

3.3 The event-driven model

Why an event-driven model?

Any execution control strategy (cyclic, time-scheduled, etc.) can be represented in terms of an event-driven model, but the converse is not necessarily true. IEC 61499 opts for the more general model in order to provide maximum flexibility and descriptive power to compliant standards and systems.

How can a change in a data value generate an event?

`E_R_TRIG` and `E_F_TRIG` function block types, defined in Annex A of IEC 61499-1, propagate an input event when the value of a Boolean input rises or falls, respectively, between successive occurrences of the input event. Instances of this type may be combined with other function blocks to produce rising- and falling-edge triggers, threshold detection events, etc.

What are event types? What are they used for?

An **event type** is an identifier associated with an event input or an event output of a function block type, assigned as part of the event input or output declaration, as described in 2.2.1.1 of IEC 61499-1. It can be used by software tools to assure that event connections are not improperly mixed, for instance, to assure that an event output that is intended to be used for initialization is not connected to an event input that is intended to be used for alarm processing.

Event types cannot be detected by execution control charts (ECCs), which control the execution of algorithms in basic function block types as defined in 2.2 of IEC 61499-1, so the type of event cannot be used to influence the processing of events in such function block types.

IEC 61499 does not define any standard event types other than the default type EVENT.

How can this model accommodate sampled-data systems?

The major issues in sampled-data systems such as those used in motion, robotic and continuous process control are

- how to achieve synchronized sampling of process or machine inputs;
- how to assure that all data has arrived in time for processing by control algorithms;
- how to assure that all outputs are present and ready for sampling before beginning the next cycle of sampling and execution.

Solutions to these problems typically require specialized communication and operating system services, which can be represented in terms of the IEC 61499 model by **service interface function blocks**. The inputs and outputs to be sampled can likewise be represented by service interface function blocks, while the algorithms to be performed will typically be encapsulated in basic function blocks. The relationships between the system services, input and output sampling, and algorithm execution can then be expressed as event connections and data connections.

What happens if a critical event is lost by the communication subsystem?

This issue is common to all distributed control systems and its solutions are well known; for example, via periodic communication, missing event detection and/or positive acknowledgement protocols. The IEC 61499 model provides for notification of abnormal operation via the `IND-`, `CNF-` and `INIT0-` service primitives and `STATUS` output of service interface function blocks.

Is there any way to distinguish between "process events" and "execution scheduling events"?

These events can be distinguished from each other by the event type mechanism. They can be used by software tools to show only the event types of interest and to restrict connections to be only among compatible event types.

How can a function block respond to faults and exceptions?

In the IEC 61499 model, faults and exceptions are modelled as event outputs and associated data outputs of service interface function blocks. These outputs can be connected to appropriate event inputs and associated data inputs of any function blocks, which must respond to the fault or exception, for example, by changing their operational modes.

Where can event handling function blocks (`E_CYCLE`, `E_RESTART`, etc.) be instantiated?

The standard does not restrict the context for instantiation of event handling function blocks or service interface function blocks in general. They can, in principle, be used wherever any other function block type may be instantiated, for instance, as components of composite function blocks or as part of a resource configuration.

Are there any mechanisms to prioritize execution of algorithms or events ?

There is just some description of priority attributes for algorithms in Annex H of IEC 61499-1; however, this description is not normative.

The rules for algorithm invocation given in 2.2.2.2 of IEC 61499-1 provide a substantial degree of control over prioritization by defining specifically the processing order of transitions and actions of an Execution Control Chart (ECC).

How can IEC 61131-3 tasks with priorities and cycling time be converted to IEC 61499?

The E_CYCLE function block is intended to be used mainly for the control of cyclic execution like the cyclic tasks of IEC 61131-3. See the previous question for a discussion of priorities.

3.4 Engineering methodologies

How can I use IEC 61499 to design and implement state-machine control?

There are various formal and informal methodologies for the design of state-machine control systems. A general outline of these methods and how IEC 61499 models and software tools can be used is as follows.

a) Define the desired sequence of operations for the controlled machine or process, as well as the abnormal sequences which may occur, if possible. This may be done informally in ordinary language, or by using more formal notations such as Petri nets or IEC 61131-3 Sequential Function Charts (SFCs).

b) Define the actuators that are to be used to implement the desired behaviour, the sensors that are to be used to determine the actual state of the controlled machine or process, and their interfaces to the state-machine controller(s). IEC 61499 service interface function block types may be used for this purpose; such type definitions will typically be provided for this purpose by the supplier of the IEC 61499-compliant sensor and actuator devices.

c) Model the behaviour of the machine or process in response to commands (events plus data) given to the actuators, and the resulting, observable time-dependent outputs (events plus data) from the sensors. This modelling may be done formally using notations such as Petri nets, or informally using simulation models (which may be implemented with appropriate IEC 61499 models).

d) Define the appropriate state-machine controllers, typically as the ECCs and algorithms of basic function block types. Methodologies for doing this step may be informal, based on engineering experience, or more formal, using for example Petri-net theory. The best method is to reuse existing, proven state-machine controllers from an IEC 61499 function block library!

e) Validate the proposed state-machine controllers versus the model of the controlled machine or process. In order to catch possible design or specification errors, this would usually be done using simulation; in addition, more formal validation methods may be used if available.

f) Finally, replace the simulated sensor and actuator interfaces with the service interfaces to the actual machine or process to be controlled. This installation should normally be carried out piecewise for testing purposes.

What is an ECC? Why should I use it and when?

An ECC is a specialized state machine to enable multiple events to trigger multiple algorithms in a basic function block type, possibly dependent on some internal state. It should be used when maximum flexibility in algorithm selection and scheduling or when a high-performance, event-driven state machine is needed. Otherwise, the mechanisms in Annex D of IEC 61499-1 for converting IEC 61131-3 function blocks to IEC 61499-style blocks can be used.

Why can a composite function block not have internal variables?

Internal variables are not required in composite function blocks, since all possible sources of data are accounted for as input or output variables of the composite function block or of its component function blocks.

Are extensible user-defined function blocks permitted?

IEC 61499 follows the usage of IEC 61131-3 and does not define a specific syntax for specification of extensible inputs and outputs of user-defined function blocks. The use of the ellipsis (...) and accompanying textual descriptions is considered adequate for the specification of extensible inputs and outputs of standard and service interface function block types.

Can alternative graphical representations be used?

Software tools can use alternative graphical, textual or tabular representations, as long as the accompanying documentation specifies an unambiguous mapping to the graphical elements and associated textual syntax defined in IEC 61499-1.

How can "trend" and "view" objects be created?

In some process control terminologies, a view is a collection of data values from various sources, arranged for remote access. This function is performed by standard IEC 61499 communication function blocks.

NOTE This usage of the term "view" differs from the well-known Model/View/Controller (MVC) model for user interface applications.

A trend is a sequence of data values from the same source. The function of collecting trend data can be implemented as an algorithm of a basic function block, and remote access to the data so collected can be provided by standard communication function blocks. See 4.1.1 and 4.1.2 for further information.

Why and when should I use the WITH construct?

When you are designing function block types, you use the WITH construct to specify an association between an event input and a set of input variables, or between an event output and a set of output variables. Subclause 2.2.2.2 of IEC 61499-1 states that this association means that the associated input variables are to be sampled at a particular transition of the ECC state machine; no specific semantics are given for the WITH associations of event outputs and output variables. It is recommended, but not required, that the following interpretations be placed on these semantics.

IEC 61499-1 states that the WITH construct is used to determine

- which input variables to sample when an event occurs at the associated event input of an instance of the type;
- which output events are used to indicate when values of associated output variables change.

In either case, it is expected that this information will be used by software tools to assist the user to ensure that

- the data used by an algorithm in one function block is consistent with the data produced by an algorithm in another function block and delivered over one or more data connections associated with one or more event connections;
- the messages transmitted over communication connections between resources in a distributed application carry consistent data and events between the function block instances in the application in the way intended by the application designer.

Are the "sequences" of service interface function blocks (SIFBs) only for documentation or can they be used for programming purposes?

The use of service sequences for SIFBs was intended for documentation purposes and especially to show a direct mapping for well-specified services which follow the ISO 8509 conventions. Even in documentation terms, however, they should be considered as imposing constraints on the externally observable operational sequences of the corresponding SIFBs. Software tools may, but are not required to, use these specifications to generate skeleton code for an implementation language such as IEC 61131-3 ST, C++ or Java.

How is IEC 61499 related to conventional methods of object-oriented development?

See Annex B.

3.5 Applications

Why does IEC 61499 not define applications as instances of application types?

It was determined that an implementation-independent specification of an application type would be identical to a composite function block type; however, this view has now evolved to that of a **sub-application type**. The configuration of applications could then be carried out according to the following process.

a) Create and interconnect one or more instances of the sub-application types (and possibly additional function block types) representing the application.

b) Create instances of the service interface function block types representing the process interfaces of the application.

c) Create appropriate event connections and data connections between the process interface function blocks and the sub-applications representing the application.

d) Remove the encapsulation around the sub-applications, exposing their component function blocks (which may also be sub-applications) as distributable elements of the application.

e) Remove the encapsulation of all of the newly exposed sub-applications, repeating as necessary.

f) Distribute the application by allocating its function block instances to appropriate resources.

g) Create and configure appropriate instances of communication function block types to maintain the event and data flows of the application.

NOTE Software tools would typically provide means of automating many of the operations in this process, especially in steps d), e) and f).

Can an application contain "sub-applications"?

Yes. See the above description as well as 2.4 of IEC 61499-1.

Can an application interface with other applications?

Not directly, since there is no **application type** within which an interface could be defined. However, applications may communicate with each other by means of **communication SIFBs**. Also, **sub-applications** may interface with each other via event connections and data connections.

How is the loading and starting of applications managed?

Software tools for this purpose will take as input a system configuration and generate sequences of commands to management function blocks to perform loading and starting of applications, either locally or remotely via communication function blocks. Details of this functionality will typically be contained in **compliance profiles** following the rules given in IEC 61499-4.

4 Examples

4.1 Applications of SIFBs

4.1.1 Views

Some system architectures define unique objects to provide specified functions. Such objects may include "transducer blocks", "views", "trends", "alerts", and remote "links". The purpose of this subclause is to illustrate the use of SIFBs to replace such specialized objects.

This example illustrates the use of communication function blocks to provide controlled remote access to real-time data and parameters of function blocks. The general approach is as follows.

a) Model the desired functionality as a function block with all accessible parameters externally visible.

b) Use appropriate SIFBs, for example, SERVER, to model the grouping and access control to parameters.

c) Encapsulate the resulting model with "invisible" contained parameters and internal access mechanisms.

Figure 1 shows a partial configuration of a resource type which provides an "operator view" via the block labelled OP_VIEW and an "engineering view" via the block labelled ENG_VIEW of the data associated with an instance of the PI_REAL composite function block example described in 2.3.1 of IEC 61499-1. The ENG_VIEW block supplies the KP and KI parameters and the HOLD variable to the PI block, while the OP_VIEW block supplies the SP variable and returns the PV variable (provided by a remote publisher via the PV_SUB block) and the XOUT variable of the PI block.

NOTE 1 The XOUT parameter would typically be connected to a PUBLISH block or a local output point to effect the desired control; this connection is not shown in this example.

NOTE 2 The naming of the communication SIFB types in this subclause shows a useful convention. The block is named XXXX_NI_NO, where XXXX is the name of the service, for example, SERVER; NI is the number of inputs, for example, 2 and NO is the number of outputs, for example, 1. In this example, the block type is SERVER_2_1; the service interface at the other end of the communication connection then has the reversed number of inputs and outputs, for example, CLIENT_1_2.

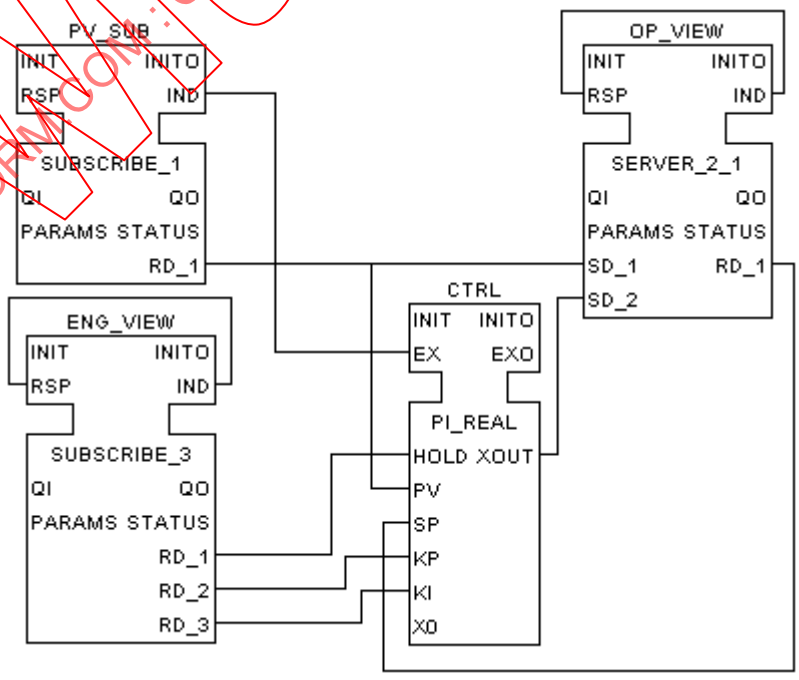


Figure 1 – Views of a PI_REAL type block

Figure 2a illustrates a partial resource configuration for a simple remote operator interface for the remote resource shown in Figure 1. An instance of a specialized version of the CLIENT function block type described in Annex F of IEC 61499-1 provides the remote interface for setting the SP (set point) of the PI algorithm, and reading the PV (process variable) input and XOUT output of the algorithm. A similar configuration could be used for an engineering interface; if these two interfaces are placed side-by-side, a display such as that in Figure 2b could be obtained.

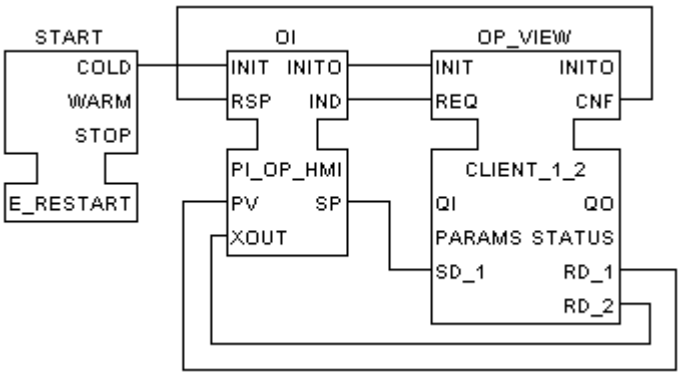


Figure 2a – Partial resource configuration

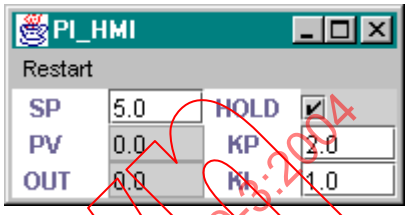


Figure 2b – Typical display

Figure 2 – Human interface

Figure 3 shows the external interface and internal function block network of the PI_OP_HMI function block type used for the simple human interface in Figure 2. In practice, a bar chart display of the three values SP, PV and OUT is typically used.

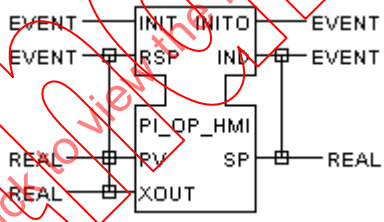


Figure 3a – External interface

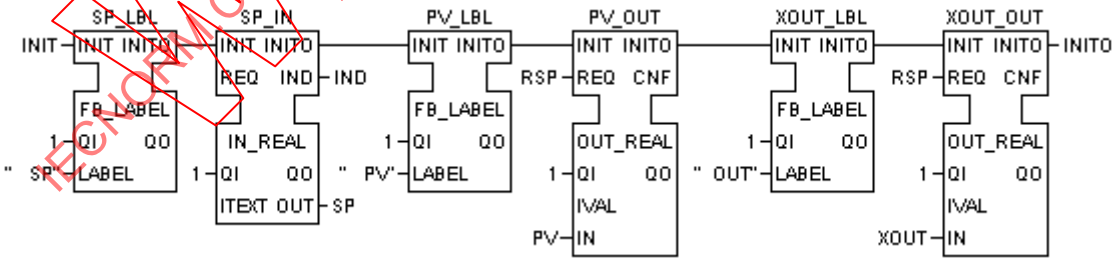


Figure 3b – Implementation

Figure 3 – Function block type PI_OP_HMI

4.1.2 Trending

Figure 4 shows a 16-sample trending function implemented in a TREND_16_REAL_VS function block. The function block provides an RD/RDO pair to assure that data being read is properly synchronized. The input data is of type REAL_VS providing both value and status of the data, as given by the declaration

```
TYPE REAL_VS:        (* REAL Value with Status byte *)
  STRUCT
    Status: BYTE;  (* Implementation dependent encoding *)
    Value: REAL;
  END_STRUCT;
END_TYPE
```

Information encoded in the `Status` byte may include

- **good (cascade)** – variable value may be used for control;
- **good (non-cascade)** – the value may be used in operation;
- **uncertain** – the variable value is suspect;
- **bad** – the variable value does not reflect the true measurement, calculation, or control value;
- additional status attributes (i.e., attributes whose interpretation depends on the main status enumerated above) and limit attributes (for example, high or low limit exceeded) may also be encoded.

Each piece of input data is stamped with the current date and time in order to account for possible non-uniform sampling intervals. The stamped data type is given by the declaration

```
TYPE STAMPED_REAL_VS:  (* Time-Stamped REAL Value with Status byte *)
  STRUCT
    Value: REAL_VS;
    Stamp: DATE_AND_TIME;
  END_STRUCT;
END_TYPE
```

Remote access to the trend values could be provided by an appropriate instance of a SERVER block as described in Annex F of IEC 61499-1.

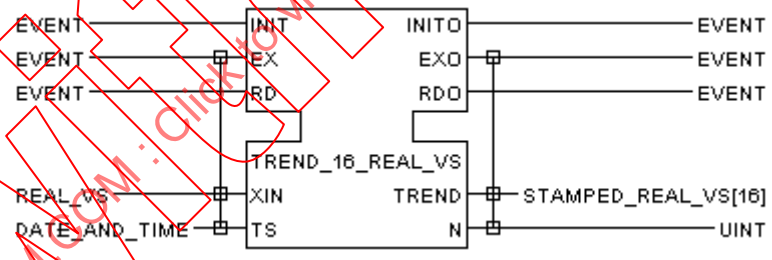


Figure 4a – External interface

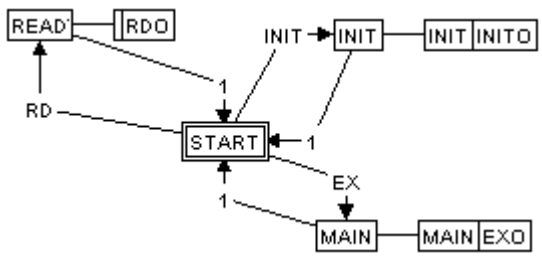


Figure 4b – ECC

Figure 4 – TREND_16_REAL_VS function block type

4.1.3 Remote sensing

Figure 5 shows a resource type, which publishes the data obtained from a thermocouple interface represented by an instance of the SIFB type TC_INTFC shown in Figure 6. The PARAMS input of the TC_INTFC block would contain data appropriate to characterize a thermocouple interface, including such items as channel number, thermocouple type, calibration data, engineering units and scaling. The RD_1 output of the TC_INTFC block would typically be of data type REAL_VS, representing a temperature in appropriate engineering units and associated status. The STATUS output would indicate conditions specific to the interface type such as open circuit, short circuit, etc.

In typical operation of this resource, the arrival of an event at the IND output of the TRIGGER block would trigger the sampling and subsequent publication of the associated data from the TC block.

A software tool as described in IEC 61499-2 would typically initiate operation of this resource via the following sequence of operations.

a) Establish a communication connection to a SERVER block connected to the MANAGER block for the resource.

b) Establish values of the PARAMS inputs of the TRIGGER, TC and PUB blocks, using WRITE or CREATE management commands.

c) Force initialization by issuing a START command as described in 3.3.2 of IEC 61499-1, which will cause an event at the WARM output of the START block to be propagated via the INIT/INITO chain of the TRIGGER, TC and PUB blocks.

d) Verify that correct initialization has occurred by using the READ command to query the values of the STATUS outputs of the TRIGGER, TC and PUB blocks, and take appropriate corrective action if necessary.

e) The software tool may also establish its own SUBSCRIBE block to the publish/subscribe channel established for the PUB block in order to monitor the status of the published variable.

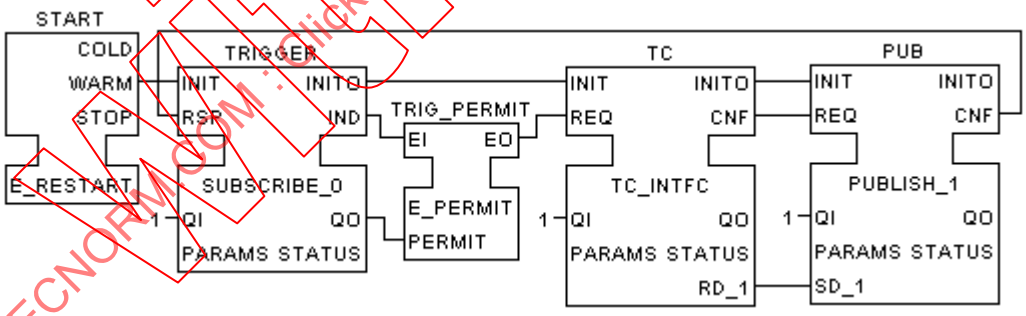
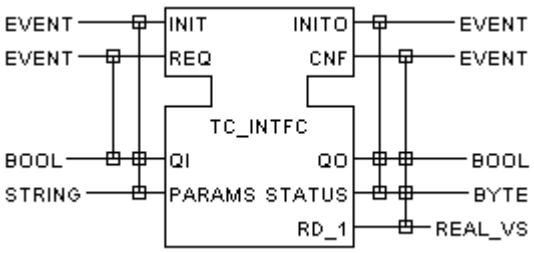


Figure 5 – Resource type TC_XMTR



NOTE See 4.1.2 for a discussion of the data type REAL_VS.

Figure 6 – SIFB type TC_INTFC

4.1.4 Remote actuation

Figure 7 shows a resource type, which subscribes to the data required to operate a valve interface represented by an instance of the SIFB type VALVE_INTFC illustrated in Figure 8. The PARAMS input of the VALVE_INTFC block contain data appropriate to characterize its operation, including such items as channel number, forward or reverse valve action, and scaling. The SD_1 input of the VALVE_INTFC block would typically be of data type REAL_VS, representing a scaled valve position and associated data status, while its STATUS output would indicate conditions such as sticking, over-travel, etc. A block PUB of type PUBLISH_1 is also provided in this resource type to publish the valve status upon occurrence of an error condition as indicated by a CNF- primitive of the VALVE_INTFC service.

In typical operation of this resource, the arrival of an event at SUB.IND would trigger the VALVE block to begin the motion of the valve to the position commanded at SUB.RD_1, followed by an event at VALVE.CNF with appropriate values at VALVE.QO and VALVE.STATUS. If an error occurs as indicated by a FALSE value of VALVE.QO, the VALVE.CNF event will be propagated to PUB.REQ via PUB_SW.EO0, causing the publication of the diagnostic information at VALVE.STATUS.

A software tool as described in IEC 61499-2 would typically initiate operation of this resource via the following sequence of operations.

a) Establish a communication connection to a SERVER block connected to the MANAGER block for the resource.

b) Establish values of the PARAMS inputs of the SUB, VALVE and PUB blocks, using WRITE or CREATE management commands.

c) Force initialization by issuing a START command, which will cause an event at the WARM output of the START block to be propagated via the INIT/INITO chain of the SUB, VALVE and PUB blocks.

d) Verify that correct initialization has occurred by using the READ command to query the values of the STATUS outputs of the SUB, VALVE and PUB blocks, and take appropriate corrective action if necessary.

e) The software tool may also establish its own SUBSCRIBE block to the publish/subscribe channel established for the PUB block in order to monitor the valve status.

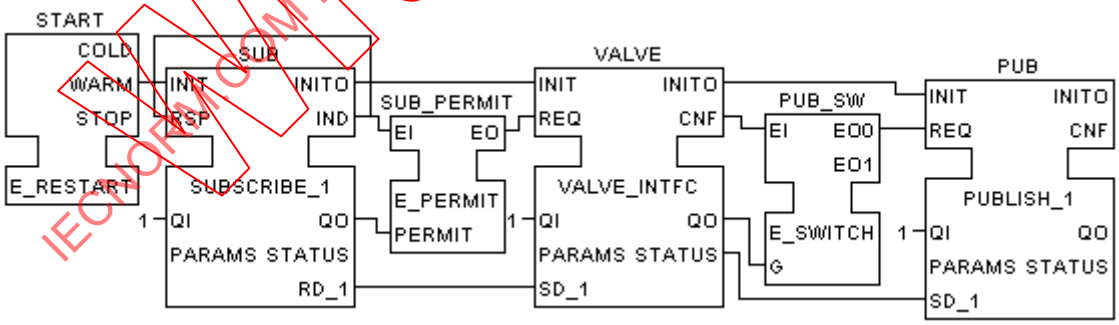
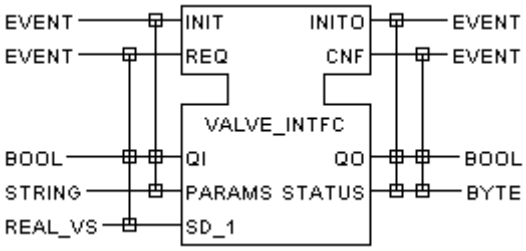


Figure 7 – Resource type VALVE_XCVR



NOTE See 4.1.2 for a discussion of the data type REAL_VS.

Figure 8 – SIFB type VALVE_INTFC

4.1.5 Remote control

Figure 9 shows a resource type which subscribes to the data required to supply the set point (SP) and process variable (PV) for the operation of an instance of the service interface function block type PID, and publishes its manipulated variable OUT, as illustrated in Figure 10. The PARAMS input of the PID block contains data appropriate to characterize its operation, including sampling time, proportional/integral/derivative constants, operating mode, etc. A block STATUS_PUB of type PUBLISH_1 is also provided in this resource type to publish the block status upon occurrence of an error condition as indicated by a CNF- primitive of the PID service.

In typical operation of this resource, the arrival of an event at PV.IND would trigger the operation of the PID algorithm, followed by an event at CTRL.CNF with appropriate values at CTRL.QO, CTRL.STATUS and CTRL.OUT. The CTRL.CNF event will be propagated to PUB.REQ or STATUS_PUB.REQ via PUB_SW.EO0 or PUB_SW.EO1 respectively, depending on whether the value of CTRL.QO is TRUE or FALSE respectively.

The set point will be changed by the arrival of an event at SP.IND and corresponding data at SP.RD_1.

A software tool as described in IEC 61499-2 would typically initiate operation of this resource via the following sequence of operations.

a) Establish a communication connection to a SERVER block connected to the MANAGER block for the resource.

b) Establish values of the PARAMS inputs of the PV, SP, CTRL, STATUS_PUB and PUB blocks, using WRITE or CREATE management commands as described in 3.3.2 of IEC 61499-1.

c) Force initialization by issuing a START command as described in 3.3.2 of IEC 61499-1, which will cause an event at the WARM output of the START block to be propagated via the INIT/INITO chain of the PV, SP, STATUS_PUB and PUB blocks.

d) Verify that correct initialization has occurred by using the READ command to query the values of the STATUS outputs of the PV, SP, STATUS_PUB and PUB blocks, and take appropriate corrective action if necessary.

e) The software tool may also establish its own SUBSCRIBE blocks to the publish/subscribe channels established for the STATUS_PUB and PUB blocks in order to monitor the PID block status and output value, respectively.

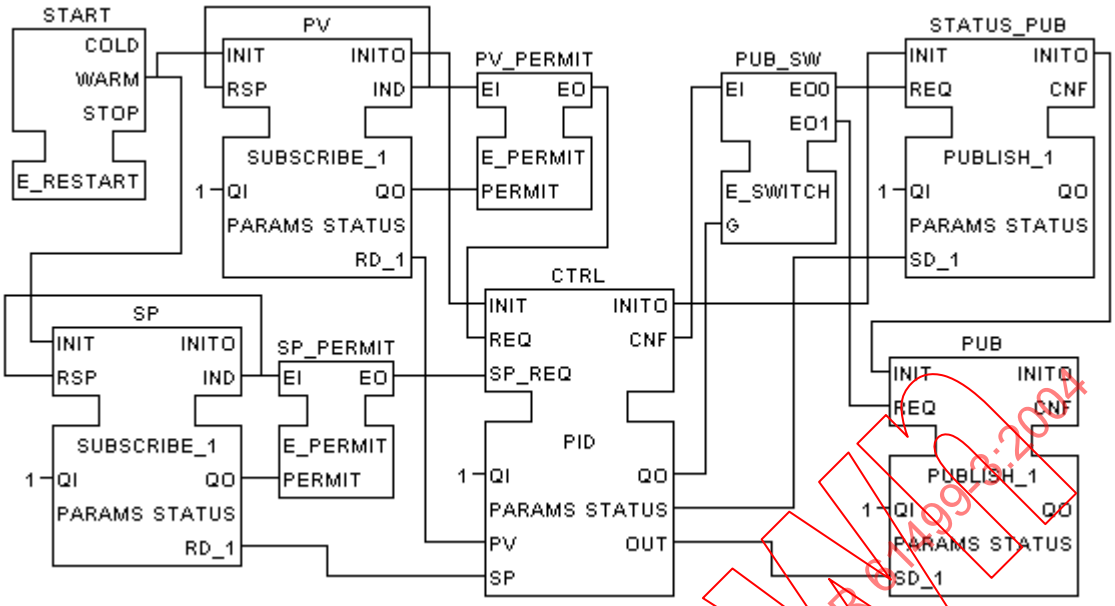
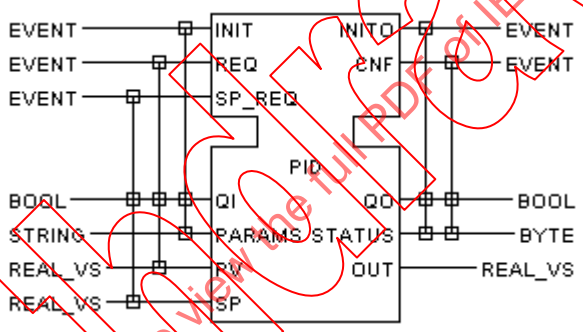


Figure 9 – Resource type PID_RSRC



NOTE 1 Specification of the internal functionality of this function block type is beyond the scope of this technical report.

NOTE 2 See 4.1.2 for a discussion of the data type REAL_VS.

Figure 10 – SIFB type PID

4.1.6 Combined control and actuation

Figure 11 shows a resource type which combines the control, actuation and diagnostic functions illustrated in Figures 9 and 10.

A software tool as described in IEC 61499-2 would typically initiate operation of this resource via the following sequence of operations.

a) Establish a communication connection to a SERVER block connected to the MANAGER block for the resource.

b) Establish values of the PARAMS inputs of the PV, SP, CTRL, STATUS_PUB, VALVE and VALVE_PUB blocks, using WRITE or CREATE management commands.

c) Force initialization by issuing a START command as described in 3.3.2 of IEC 61499-1, which will cause an event at the WARM output of the START block to be propagated via the INIT/INITO chain of the PV, SP, CTRL, STATUS_PUB, VALVE and VALVE_PUB blocks.

d) Verify that correct initialization has occurred by using the READ command to query the values of the STATUS outputs of the PV, SP, CTRL, STATUS_PUB, VALVE and VALVE_PUB blocks, and take appropriate corrective action if necessary.

e) The software tool may also establish its own SUBSCRIBE blocks to the publish/subscribe channels established for the STATUS_PUB and VALVE_PUB blocks in order to monitor the status of the PID and VALVE blocks, respectively.

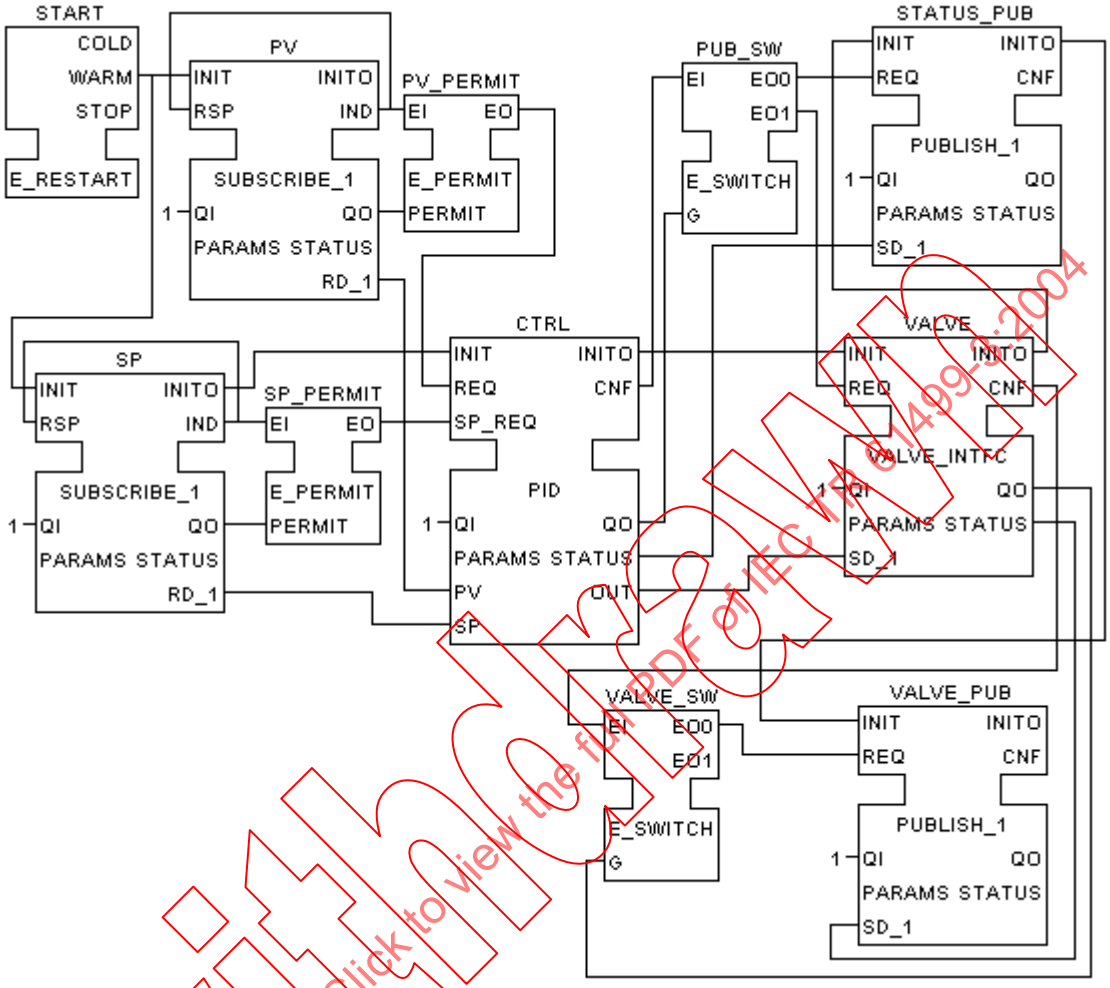


Figure 11 – Resource type PID_VALVE

4.2 System configuration

Figure 12 illustrates how some of the elements defined in 4.1 may be combined into a simple system configuration. Features of interest in this configuration are as follows.

a) For the purposes of declaration and parameterization, communication connections may be modelled as function blocks which exist at the system or resource level, depending on whether the connection end points are in different devices or the same device, respectively.

b) In this example, it is assumed that either some sort of "directory service" exists that enables the service interfaces at the connection end points to locate the communication connections by a name given at the PARAMS input of the corresponding service interface function block, or that a software tool as described in IEC 61499-2 is able to use this information to set up appropriate communication connections in an implementation dependent manner. This is the only additional information that needs to be included in the resource configurations in order to integrate this application.

c) A simple periodic scheduling scheme can be set up for the system by adding an E_CYCLE block to issue periodic events, as shown in the configuration for resource TC.LAS. The resulting timing diagram is shown in Figure 13, assuming a function block execution time of 10 ms, data transmission times of 5 ms, and valve and thermocouple interface execution times of 5 ms.

A software tool as described in IEC 61499-2 would typically initiate operation of this system via the following sequence of operations.

a) Establish initial values for parameters in the TC.TCX and VALVE.V resources using the procedures described for the TC_XMTR and PID_VALVE resource types in 4.1.2 and 4.1.5, respectively. In this example, this would include, among other actions, setting a value of "CLK_CNXN" for TC.TCX.TRIGGER.PARAMS and a value of "PV.CNXN" for TC.TCX.PUB.PARAMS and VALVE.V.PV.PARAMS.

b) Establish a communication connection to a SERVER block connected to the MANAGER block for the TC device and use the CREATE management command to add the LAS resource and the CLK_CNXN block to the TC device.

c) Establish a communication connection to a SERVER block connected to the MANAGER block for the TC.LAS resource and then

   1) use CREATE management commands to populate the resource with the MAIN.CLK and the CLK_PUB blocks;

   2) use CREATE commands to establish the event and data connections within the resource;

   3) use WRITE or CREATE commands to establish parameter values for the blocks in the resource.

   NOTE The sequence of commands in the steps described above may be derived directly from the sequence of textual declarations of the associated device and resource configurations, as illustrated in Annex B.

d) Force initialization by issuing START commands to the appropriate devices through their management blocks. The exact order of the START commands will depend upon the required initialization services of the communication function blocks used.

e) The software tool could also create an **operator interface application** by establishing its own communication service interfaces and operator interface blocks for monitoring and operating the various resources, establishing appropriate communication connections and parameterizing the communication service interfaces available in the various resources for this purpose.

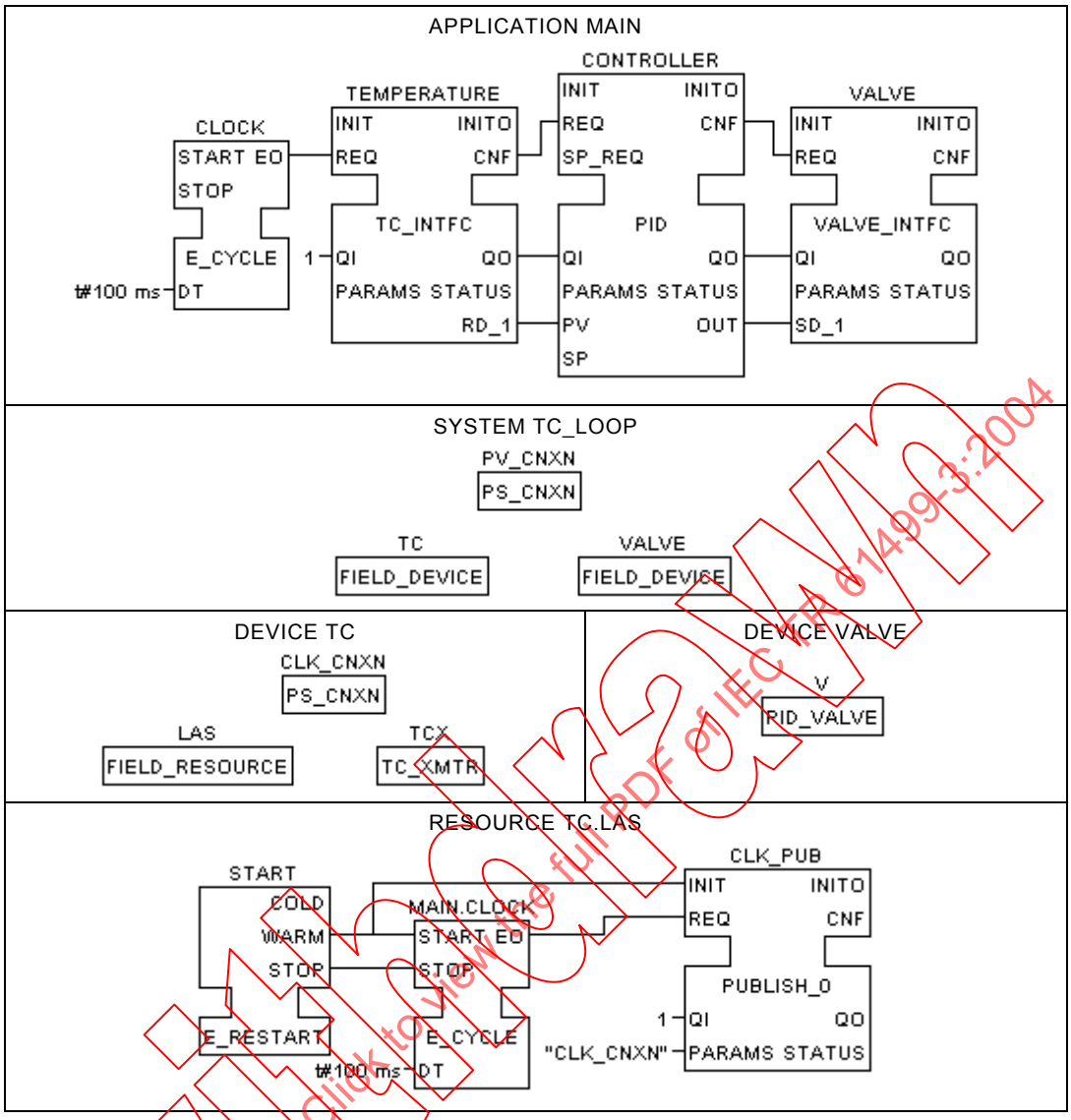


Figure 12 – System configuration TC_LOOP

CLK_CNXN					
MAIN.TEMPERATURE					
PV_CNXN					
MAIN.CONTROLLER					
MAIN.VALVE					
elapsed time, ms	**5**	**10**	**15**	**25**	**30**

Figure 13 – Example system timing

4.3 Use of communication function blocks

When an application is distributed across resources, those data and event connections of the application that cross resource boundaries must be mapped onto communication connections among the resources. In order to assure proper synchronization of application operations, a unidirectional communication connection comprising an instance of the PUBLISHER function block type and one or more instances of the SUBSCRIBER function block type described in F.2.1 of IEC 61499-1 must be provided for each event output and its associated data outputs whose connections cross resource boundaries; or, if pairwise linking of two event connections can be identified, a bidirectional communication connection comprising a CLIENT/SERVER pair, as described in F.2.2 of IEC 61499-1, may be used.

The required communication function blocks may be generated in several different ways, for example:

a) the application developer may specify them explicitly, especially for operating and monitoring purposes;

b) software tools may generate them automatically for any connection across resource boundaries.

4.4 Contained variables in process control function blocks

For some application domains such as those addressed by the IEC 61804 series, **contained variables** are defined as variables whose values are configured, set by an operator, higher level device, or calculated. Access to contained variables can be specified by **access paths** as illustrated in the following example of parsing the access path PATHA in the device type MOTOR_CONTROL.

Type declarations	Parse of PATHA in MOTOR_CONTROL

TYPE MEASUREMENT: STRUCT
  V: VOLTS;
  A: AMPERES;
END_STRUCT; END_TYPE
 	
TYPE VOLTS: STRUCT
  PhsA: REAL;
  PhsB: REAL;
END_STRUCT; END_TYPE
 	
FUNCTION_BLOCK PROTECTOR
...
VAR
  MX: MEASUREMENT;
  ...
END_VAR
...
END_FUNCTION_BLOCK
 	
FUNCTION_BLOCK BREAKER
...
FBS
  TOC: PROTECTOR;
  ...
END_FBS
...
END_FUNCTION_BLOCK
 	
RESOURCE_TYPE MOTOR_CONTROL
...
FBS
  BreakerA: BREAKER;
  ...
 	
END_FBS
...
VAR_ACCESS
  PATHA: BreakerA.TO3.MX.A.PhsA;
  ...
END_VAR
...
END_RESOURCE
	BreakerA -- fb_instance_name
	  .TOC -- fb_instance_name
	  .MX -- variable_name
	  .A -- field_selector
	  .PhsA -- field_selector

NOTE Suppressed detail in the above example is indicated by the ellipsis (...)	

4.5 Use of adapter interfaces to implement object-oriented concepts

Figure 14a shows an adapter type which can be used to provide **polymorphism** in the object-oriented programming sense. This adapter defines an interface for any typed function of two variables of type INT.

As shown in Figure 14b, the normal operation of this interface may be considered to consist of sending a **message** from an acceptor function block, which may be considered a **client** of the mathematical function, to a provider function block, which may be considered a **server** providing the mathematical function. This is followed by sending another message from the server to the client consisting of the results of the function evaluation. Figures 14c and 14d respectively represent the messages sent when evaluation of the function is inhibited and when it results in an error, respectively. Detailed textual declarations of these service sequences are given in Figure 14e.

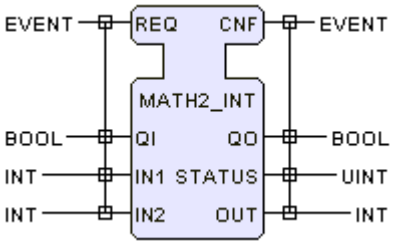


Figure 14a – Interface

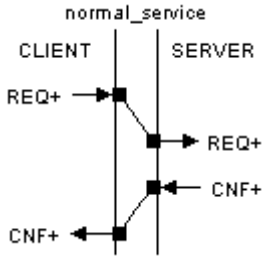


Figure 14b – Normal service sequence

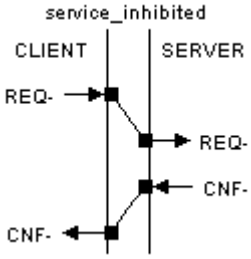


Figure 14c – Service inhibited

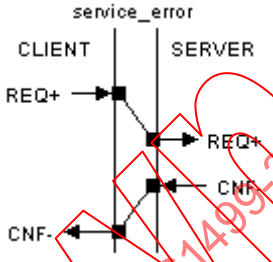


Figure 14d – Service error

```
SERVICE CLIENT/SERVER

SEQUENCE normal_service
    CLIENT.REQ+(IN1,IN2) -> SERVER.REQ+(IN1,IN2);
    SERVER.CNF+(OUT) -> CLIENT.CNF+(OUT);
END_SEQUENCE

SEQUENCE service_inhibited
    CLIENT.REQ-() -> SERVER.REQ-();
    SERVER.CNF-(STATUS) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE service_error
    CLIENT.REQ+(IN1,IN2) -> SERVER.REQ+(IN1,IN2);
    SERVER.CNF-(STATUS) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

END_SERVICE
```

Figure 14e – Textual declarations of service sequences

Figure 14 – Polymorphic adapter type declaration

Different instances of the "client" acceptor function block type shown in Figure 15 may exhibit different behaviour, depending on the functionality that is plugged into the socket S1. In object-oriented programming terms, this polymorphic behaviour is provided through "inheritance by parts" rather than "inheritance by descent".

NOTE 1 The values of QO and STATUS are passed directly from the adapter instance S1 to the outputs of the "parent" function block since the particular implementation of FB_ADD_INT shown does not provide these values.

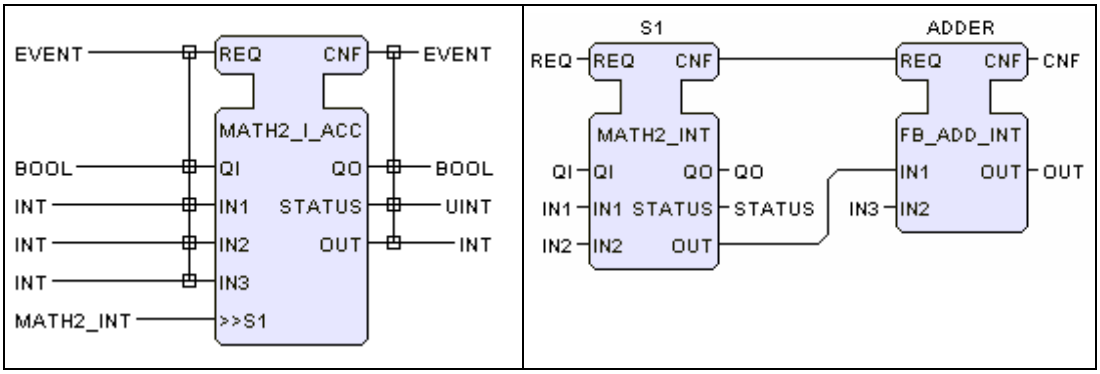


Figure 15 – Polymorphic acceptor ("client") function block type

Figures 16a and 16b show a "server" provider function block type which provides INT multiplication through a MATH2 interface, while Figures 16c and 16d show the provider of an INT division function.

NOTE 2 In the function block body shown in Figure 16b, the value of P1.QI is passed directly to P1.QO, and the value of P1.STATUS is set to 0 (false) since the particular implementation of FB_ADD_MUL shown does not provide these values.

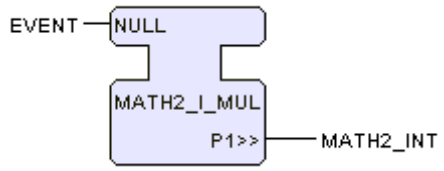


Figure 16a – Multiplication provider interface

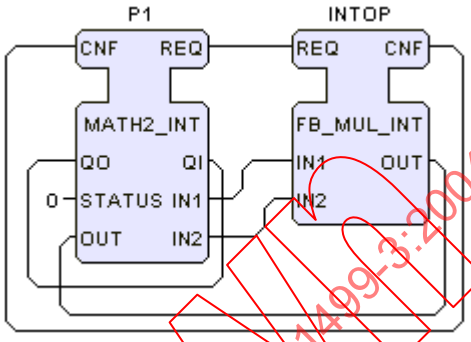


Figure 16b – Multiplication provider body

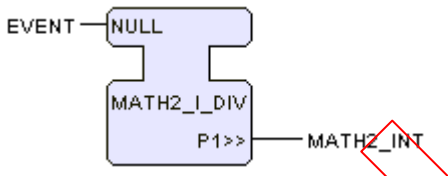


Figure 16c – Division provider interface

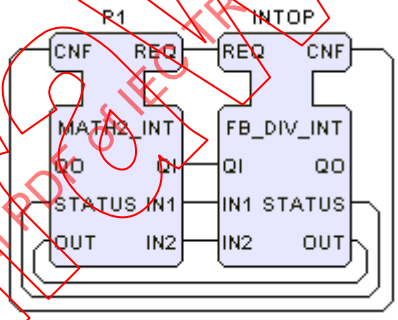


Figure 16d – Division provider body

Figure 16 – Provider ("server") function block types

Figure 17 shows a resource configuration for testing an instance named ACC of the MATH2_I_ACC type. Providers for the multiply and divide operations are given as instances of the MATH2_I_ADD and MATH2_I_MUL, respectively.

Figure 18a shows the result of a test of the configuration with the MATH2_I_DIV instance (DIV_PROV) acting as the provider, giving the result OUT = IN1/IN2 + IN3. The top half of this example shows normal operation and the bottom half shows QO=FALSE with STATUS=4, indicating an invalid value of OUT due to division by zero as specified in Annex D of IEC 61499-1. Figure 18b shows the result of a test of the configuration with the MATH2_I_MUL instance (MUL_PROV) acting as the provider, giving the result OUT = IN1*IN2 + IN3. The bottom half of this figure shows the result of operation with QI=FALSE; in this case QO=FALSE but no reason is given in the STATUS output since this output is not given by the particular implementation of the MATH2_I_MUL block. In the case of the MATH2_I_DIV block, the STATUS output would have a value of 1 as specified in Annex D of IEC 61499-1.

Management function blocks may be utilized to modify the behaviour of a polymorphic acceptor instance, for example, by moving the adapter connection start point from DIV_PROV.P1 to MUL_PROV.P1 in Figure 17.

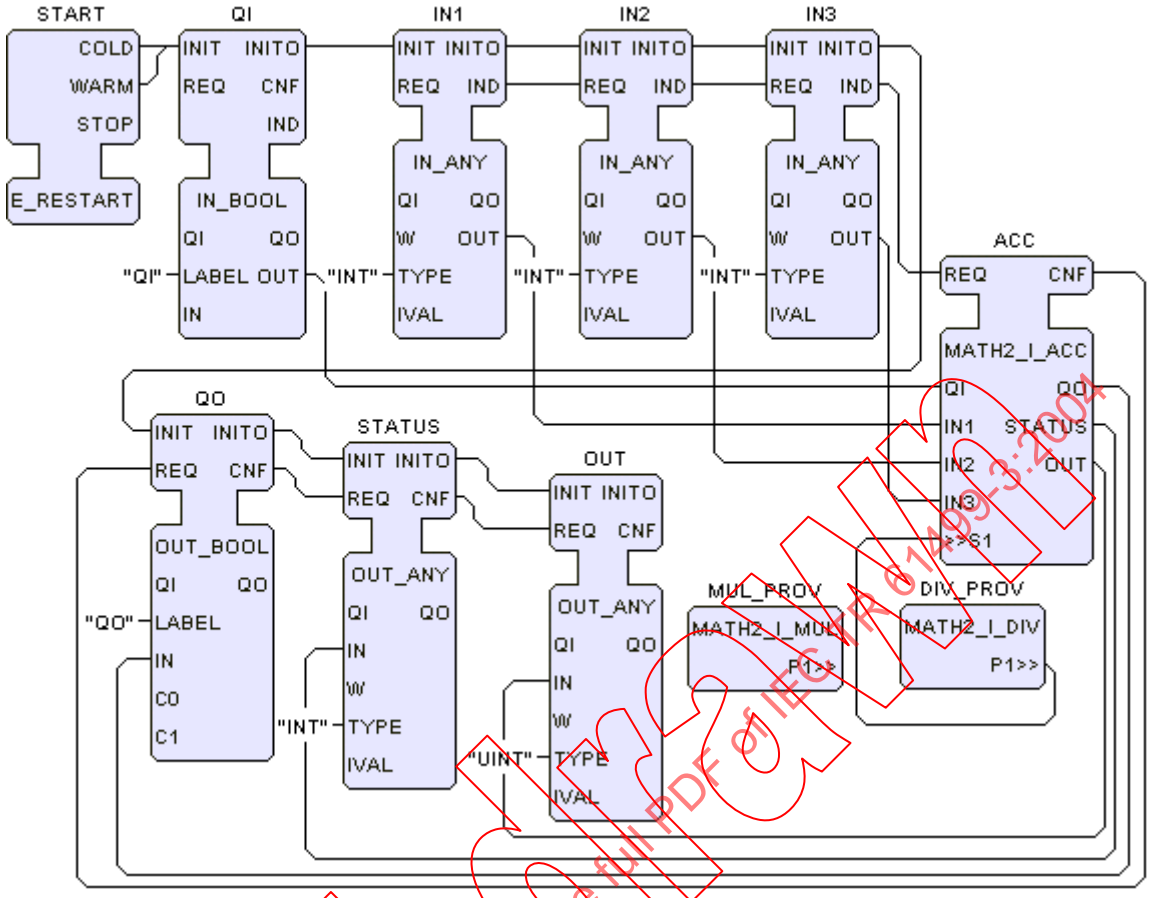


Figure 17 – Resource configuration for testing adapter interfaces

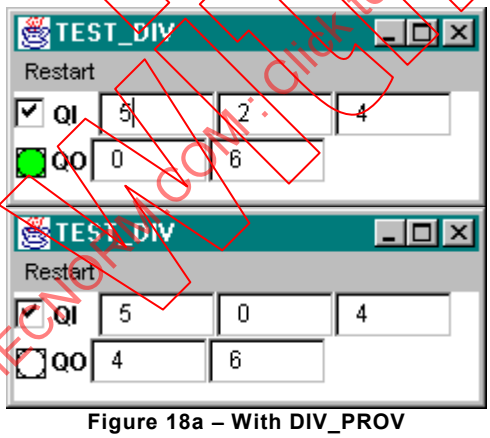


Figure 18a – With DIV_PROV

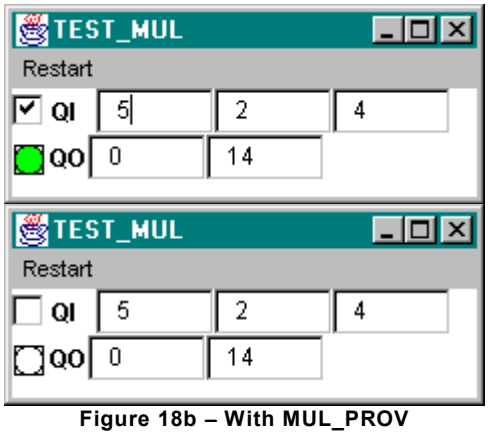


Figure 18b – With MUL_PROV

Figure 18 – Test results

4.6 Initialization algorithms

Subclause 2.2.2.1 of IEC 61499-1 states:

"The function block type may also specify an initialization algorithm to be performed upon the occurrence of an appropriate event, for example [an] `INIT` algorithm…. An application can then specify the conditions under which this algorithm is to be executed, for example by connecting an output of an instance of the `E_RESTART` type defined in Annex A to an appropriate event input….".

Figure 19a shows the configuration of a Human/Machine Interface (HMI) resource illustrating the above principles. This resource contains instances of three HMI service interface types:

- the IN_EVENT type, which provides a push-button for input of events;
- the CHOICE_IN type, which provides a WSTRING output from a drop-down list of choices;
- the OUT_ANY type, which provides a text field for output of the literal value of an arbitrary data type.

The operation of the initialization algorithm of each of these types upon the occurrence of an INIT+ service primitive (i.e., an occurrence of an event at the INIT input when the value of the QI input is TRUE) is as follows.

a) If the graphical user interface (GUI) element does not exist, it is created and added to the HMI panel.

b) The GUI element contents are then initialized from the input parameters of the block. The particular initializations are:

   - the IN_EVENT label is initialized from the LABEL input;
   - the OUT_ANY width in characters is set from the W input, its initial content is set from the IVAL input, and any ambiguity in the actual data type is resolved by the contents of the TYPE input (for instance, "UINT").
   - the choices presented by the CHOICE_IN type are given as comma-separated substrings of the CHOICES input, and the initial choice presented is the first element of the list.

c) An INITO+ service primitive (an event at the IND output accompanied by a QO value of TRUE) is emitted.

The normal operation of these blocks is as follows.

- IN_EVENT: When the user clicks on the button, an IND+ service primitive occurs.
- CHOICE_IN: The occurrence of a REQ+ input service primitive or user selection of a value causes OUT to take on the value of the current choice, and an IND+ then occurs.
- OUT_ANY: The occurrence of a REQ+ input service primitive or user-pressed **Enter** key causes OUT to take on the currently entered, and a CNF+ then occurs.

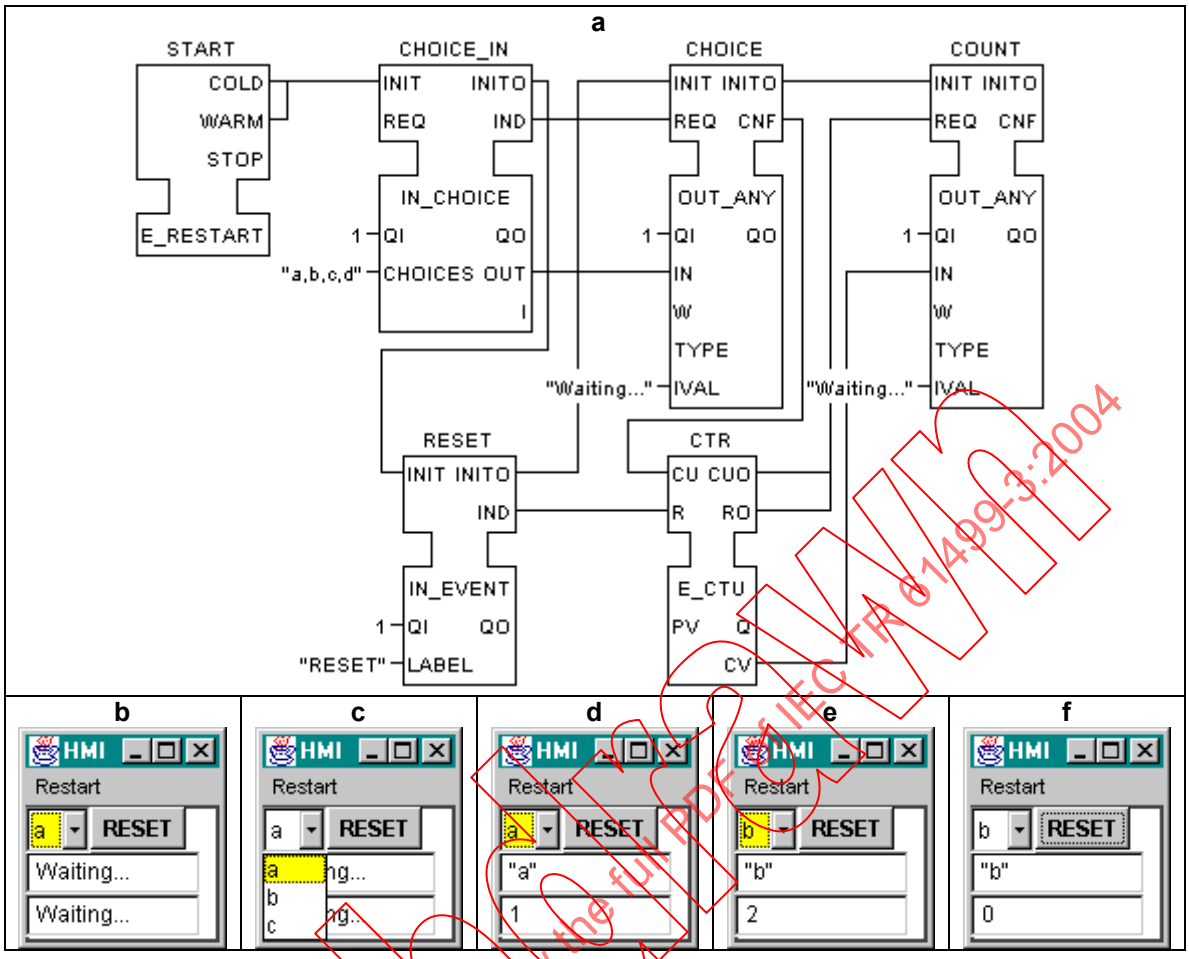


Figure 19 – HMI example

The remaining entries in Figure 19 illustrate the operation of this HMI as follows.

a) The initial situation upon COLD start or WARM restart of the resource.

b) Initial data selection from the drop-down list.

c) State of the HMI following initial selection.

d) State of the HMI following selection of the "b" choice.

e) State of the HMI following a click of the RESET button.

5 State chart implementation with ECCs

The implementation of state-machine control via ECCs can be simpler than more general-purpose state-machine models. For instance, the state chart shown in Figure 20 for (simplified) control of a hypothetical Video Cassette Recorder (VCR) motor contains the following features not included in the ECC construct:

- a "superstate" RUNNING containing two "substates" FWD and REV;
- a "history" node to which return can be made from an external state; for instance, if the RUN state is active upon the occurrence of a PAUSE event, the RUN state will be re-activated upon return from the PAUSE state in response to a RESUME event.

NOTE An alternative approach to implementing the state machine in Figure 20, utilizing a separate function block instance for each of the super-states RUNNING, STOPPED and PAUSED, is given in "An Object Oriented Approach to Generate Executable Code from the OMT-based Dynamic Model", *Journal of Integrated Design and Process Science*, December 1998, Vol.2, No. 4.

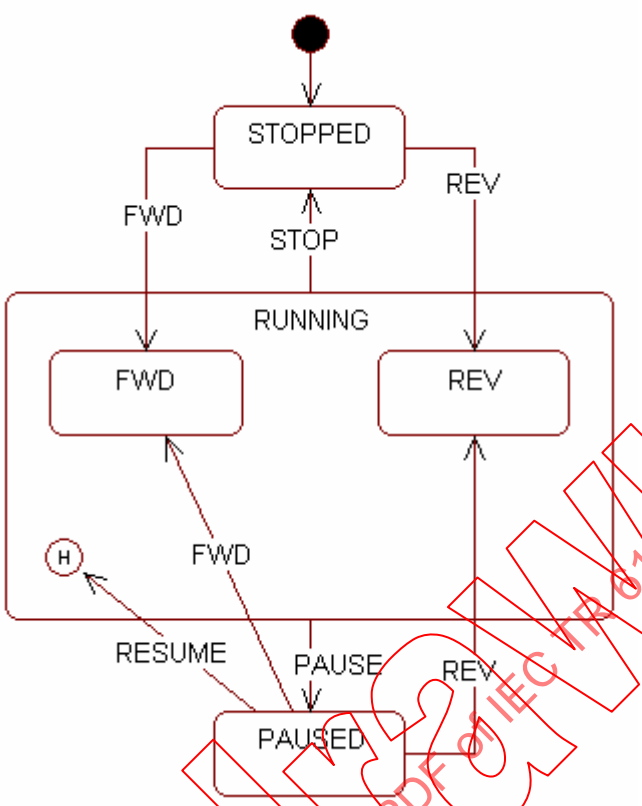


Figure 20 – State machine for hypothetical VCR motor control

It is important to note that **the state machine notation in Figure 20 is a design notation, whereas the ECC is an implementation notation.** Figure 21 presents the interface and ECC of a basic function block **implementing** the VCR motor control whose **design** is expressed in Figure 20. The ECC retains the same states as the original state chart and uses an internal variable `WAS_FWD` to save the state of the motor while pausing. This saved state is then used to restore the proper motor state when resuming operation. The declarations used to effect this functionality are:

```
VAR
  WAS_FWD: BOOL; (* Motor State History *)
END_VAR
ALGORITHM STOP IN ST:
  MTR_FWD:=FALSE;
  MTR_REV:=FALSE;
END_ALGORITHM
ALGORITHM FWD_M IN ST:
  WAS_FWD:=TRUE;
END_ALGORITHM
ALGORITHM REV_M IN ST:
  WAS_FWD:=FALSE;
END_ALGORITHM
ALGORITHM RUN IN ST:
  MTR_FWD:=WAS_FWD;
  MTR_REV:=NOT WAS_FWD;
END_ALGORITHM
```

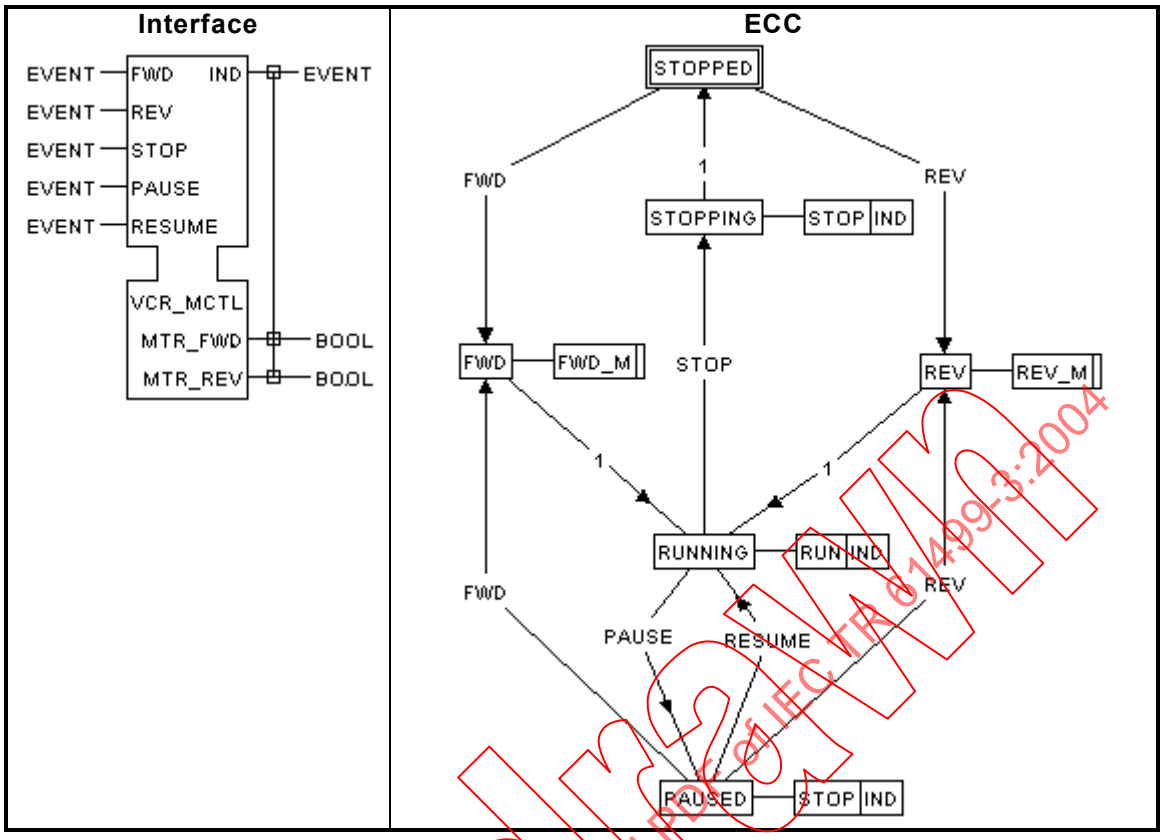


Figure 21 – Basic function block implementation of state chart

6 Device and resource management

6.1 Distributed management application

As shown in Figure 22, the functions of device management may be modelled as a management application in which a "management client" provided by a software tool and a "management server" located within the managed device. These are modelled as instances of the `DM_CLT` and `DEV_MGR` types, respectively.

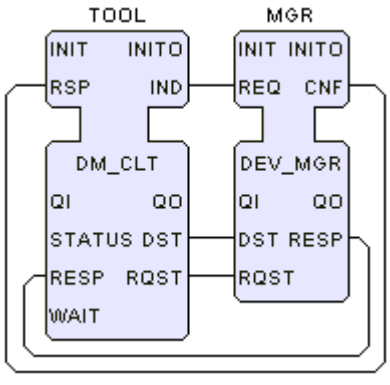


Figure 22 – Device management application

Since the software tool is typically located in a separate device from the one being managed, the management application will typically be distributed. Communication service interfaces can be used to implement the necessary communications between a "remote device proxy" in the software tool and a management resource in the remote device. Such elements can be constructed as instances of the `RMT_PRXY` and `DM_RES` types shown in Figures 23 and 24, respectively.

NOTE 1 The interface of the software tool with the management application is represented by an instance of the DM_CLT type.

NOTE 2 The interface and functionality of the DM_KRNL type is described in 5.2.

NOTE 3 Suppliers of devices are responsible for providing the equivalent of the value of the MGR_ID input of their instance of the DM_RES type shown in Figure 27. For instance, this may be the value of the host port element defined in IETF RFC 1630 to be used for access to the device management services, such as "localhost:61499". This value may be defined as part of a library element file for the device type or may be configured through some means beyond the scope of IEC 61499, for instance, via a local serial port or configuration file.

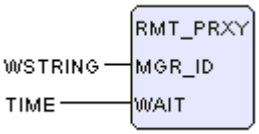


Figure 23a – Interface

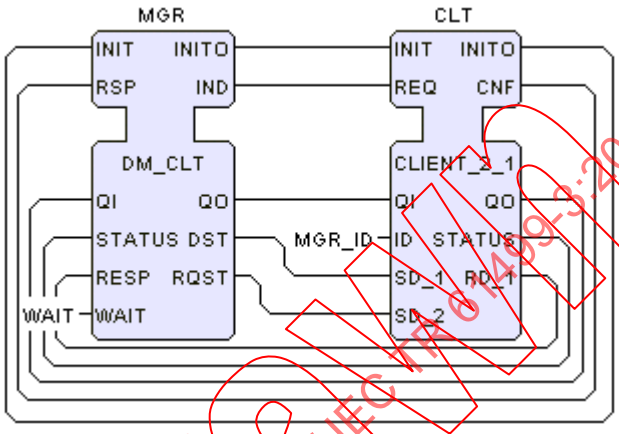


Figure 23b – Body

Figure 23 – Remote device proxy

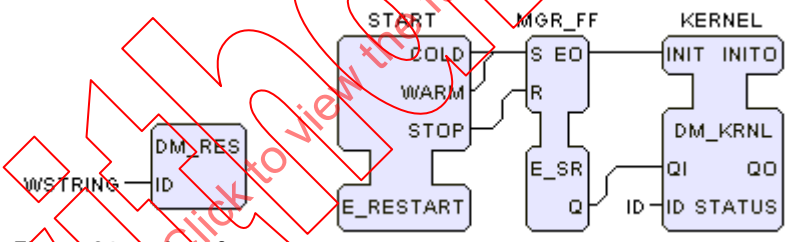


Figure 24a – Interface

Figure 24b – Body

Figure 24 – Device management resource

6.2 Device management function blocks

As shown in Figure 26, the DM_KRNL function block shown in Figure 25 provides both the communication service interface (an instance of the SERVER_1_2 type) and the device management service interface (an instance of the DEV_MGR type) to support the distributed management application illustrated in Figure 22, in conjunction with the proxy device shown in Figure 23, which is used by an appropriate software tool.

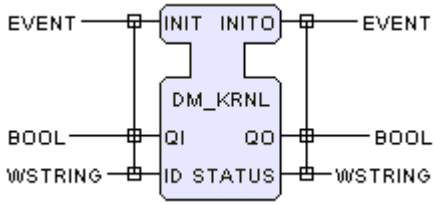


Figure 25a – Interface

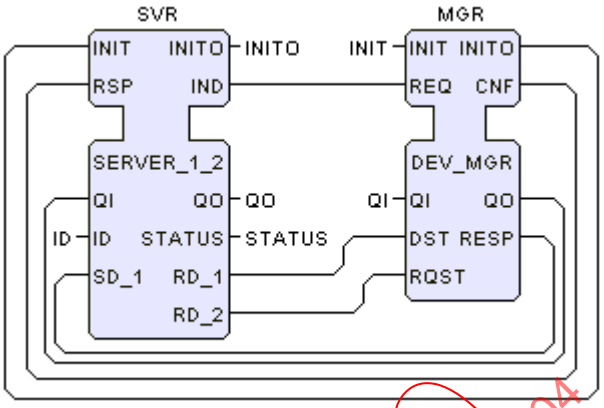


Figure 25b – Body

Figure 25 – Device management kernel

Device management services are provided by an instance of the `DEV_MGR` type shown in Figure 26. The types and semantics of the inputs and outputs of this type are identical to the correspondingly named inputs and outputs of the MANAGER type defined in IEC 61499-1, with the following extensions.

a) The `DST` input designates the destination of the `RQST` input as follows:

- a value of `""` (the empty string) designates the device;
- a value containing an IEC 61131-3 identifier designates a resource within the device;
- a value containing a sequence of IEC 61131-3 identifiers separated by periods (the `'.'` character) indicates a resource in a containment hierarchy of resources, with the leftmost identifier corresponding to the outermost resource and the rightmost identifier corresponding to the innermost resource.

EXAMPLE 1 A `DST` value of `'RES1'` indicates that the `RQST` input is destined for a resource named `RES1` contained in the managed device.

EXAMPLE 2 A `DST` value of `'MOTOR1.WINDING2'` indicates that the `RQST` input is destined for a resource named `WINDING2` contained in a resource named `MOTOR1` which is contained in the managed device.

b) The `RQST` input and `RESP` outputs are encoded according to the `Request` and `Response` elements, respectively, of the XML DTD given in 5.3. The semantics of these elements are defined in 5.4.

c) As illustrated in Figure 26, a `REQ+` primitive input always results in a `CNF+` primitive output, since the actual result including failure conditions is encoded in the `RESP` output. Similarly, a `REQ-` input always results in a `CNF-` output, since no management operation is attempted in this case. In particular, this means that, in an instance of the `DM_KRNL` function block type shown in Figure 25, an `IND-` primitive from the communication service interface will neither cause a management operation to be performed, nor will a response message be generated.

The sequences of service primitives for device management are shown in Figure 26. The object denoted "manager" in these service sequences is an instance of class **FBManager** described in Annex C of IEC 61499-1. This is the manager of the device or a contained resource depending on the value of the `DST` input as explained above.

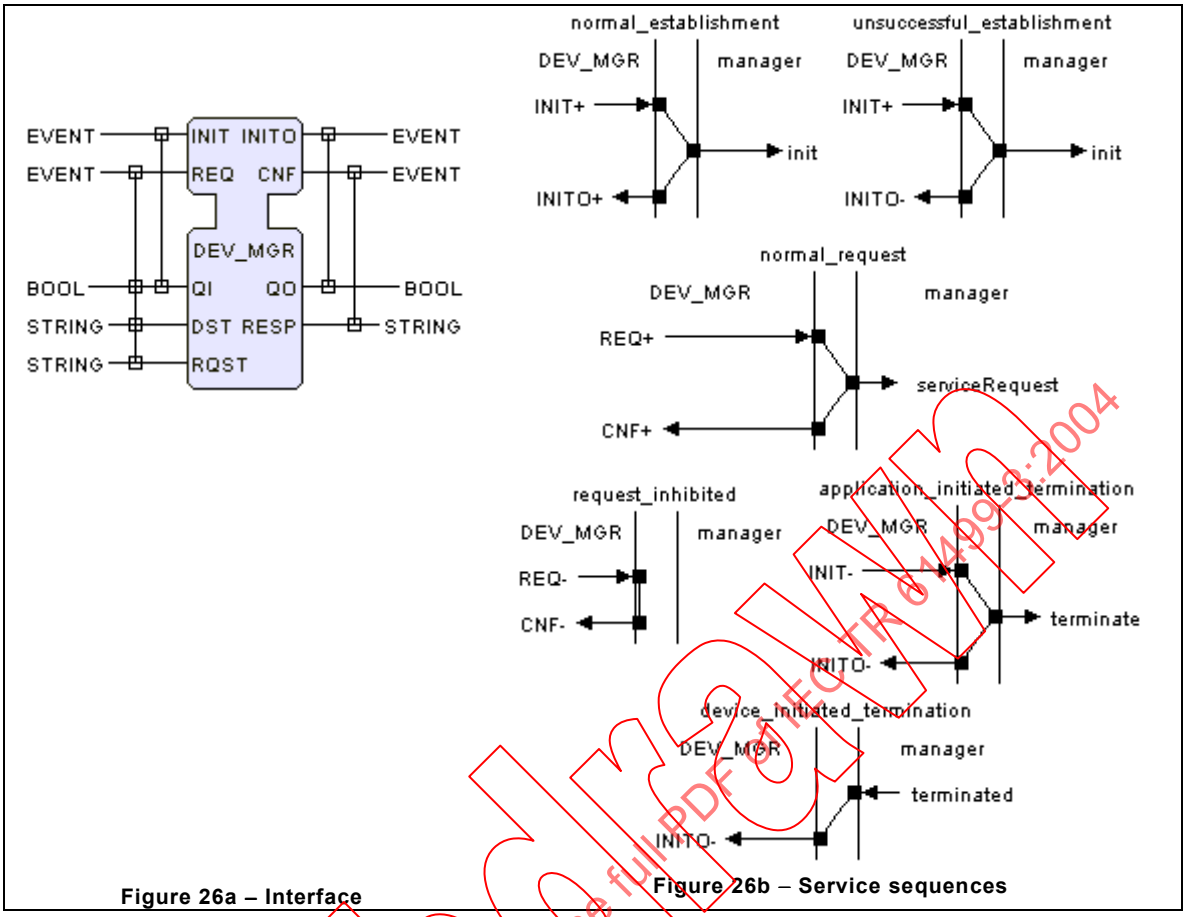


Figure 26a – Interface

Figure 26b – Service sequences

Figure 26 – Device management service interface

6.3 FBMGT Document Type Definition (DTD)

The `Request` and `Response` elements defined in the `FBMGT` DTD shown in Table 1 represent the XML syntax for the `RQST` input and `RESP` output, respectively, of the `DEV_MGR` function block type shown in Figure 26. Explanations of the elements of this DTD and (where applicable) references to the formal syntax for their attributes, are given in Table 2. Allowable combinations of elements, and constraints on their usage, are as described for various device classes in IEC 61499-4.

NOTE To provide compact messaging, the `prolog` and `Misc*` components used in the XML `document` production are not used in FBMGT messages since these components are implicit in the management context; thus, only the `Request` or `Response` element is transmitted as the management message.

Table 1 – FBMGT DTD

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!-- REQUEST elements -->
<!ELEMENT Request (FB|Connection|FBType|AdapterType|DataType)>
<!ATTLIST Request
  ID CDATA #REQUIRED
  Action (CREATE|DELETE|START|STOP|KILL|QUERY|READ|WRITE) #REQUIRED >
```

```
<!-- RESPONSE elements -->

<!ELEMENT Response  (FB | Connection+ | FBType | AdapterType | DataType |
NameList | FBList |EndpointList | FBStatus)?>
<!ATTLIST Response
  ID CDATA #REQUIRED
  Reason (NOT_READY | UNSUPPORTED_CMD | UNSUPPORTED_TYPE |NO_SUCH_OBJECT |
INVALID_OBJECT| INVALID_OPERATION | INVALID_STATE | OVERFLOW) #IMPLIED >
```

Table 1 – FBMGT DTD

```
<!ELEMENT NameList (#PCDATA)>
<!ELEMENT FBList (#PCDATA)>
<!ELEMENT EndpointList (#PCDATA)>

<!ELEMENT FBStatus EMPTY>
<!ATTLIST FBStatus
  Status (INITIALIZED|WAITING|EVALUATING|PROCESSING|STOPPED|KILLED)
#REQUIRED >
```

```
<!-- Common elements -->
<!ELEMENT ByteData (#PCDATA)>

<!ELEMENT VersionInfo EMPTY>
<!ATTLIST VersionInfo
  Organization CDATA #REQUIRED
  Version CDATA #REQUIRED
  Date CDATA #REQUIRED >

<!ELEMENT FB EMPTY>
<!ATTLIST FB
  Name CDATA #REQUIRED
  Type CDATA #REQUIRED >

<!ELEMENT Connection EMPTY>
<!ATTLIST Connection
 Source CDATA #REQUIRED
 Destination CDATA #REQUIRED >

<!ELEMENT VarDeclaration EMPTY>
<!ATTLIST VarDeclaration
  Name ID #REQUIRED
  Type CDATA #REQUIRED
  ArraySize CDATA #IMPLIED
  InitialValue CDATA #IMPLIED >
```

```
<!-- FBType elements -->
<!ELEMENT FBType (VersionInfo,InterfaceList,ByteData?) >
<!ATTLIST FBType
  Name CDATA #REQUIRED >

<!ELEMENT InterfaceList
(EventInputs?,EventOutputs?,InputVars?,OutputVars?, Sockets?, Plugs?)>

<!ELEMENT EventInputs (Event+)>
<!ELEMENT EventOutputs (Event+)>
<!ELEMENT InputVars (VarDeclaration+)>
<!ELEMENT OutputVars (VarDeclaration+)>
<!ELEMENT Sockets (AdapterDeclaration+)>
<!ELEMENT Plugs (AdapterDeclaration+)>

<!ELEMENT Event EMPTY>
<!ATTLIST Event
  Name ID #REQUIRED
  Type CDATA #IMPLIED
  With CDATA #IMPLIED >

<!ELEMENT AdapterDeclaration EMPTY>
<!ATTLIST AdapterDeclaration
   Name ID #REQUIRED
  Type CDATA #REQUIRED >
```

```
<!-- AdapterType elements -->
<!ELEMENT AdapterType (VersionInfo,InterfaceList,ByteData?)>
<!ATTLIST AdapterType
  Name ID #REQUIRED >
```

Table 1 – FBMGT DTD

```
<!-- DataType elements -->
<!ELEMENT DataType (VersionInfo,ASN1Tag, (DirectlyDerivedType
|EnumeratedType|SubrangeType|ArrayType|StructuredType),ByteData?)>
<!ATTLIST DataType
  Name ID #REQUIRED >

<!ELEMENT ASN1Tag EMPTY>
<!ATTLIST ASN1Tag
  Class (UNIVERSAL | APPLICATION | CONTEXT | PRIVATE) #IMPLIED
  Number CDATA #REQUIRED >

<!ELEMENT DirectlyDerivedType EMPTY>
<!ATTLIST DirectlyDerivedType
  BaseType (BOOL | SINT | INT | DINT | LINT | USINT | UINT | UDINT | ULINT
| REAL | LREAL | TIME | DATE | TIME_OF_DAY | DATE_AND_TIME | STRING | BYTE
| WORD | DWORD | LWORD | WSTRING) #REQUIRED
  InitialValue CDATA #IMPLIED >

<!ELEMENT EnumeratedType (#PCDATA)>
<!ATTLIST EnumeratedType
  InitialValue CDATA #IMPLIED >

<!ELEMENT SubrangeType (Subrange)>
<!ATTLIST SubrangeType
  BaseType (SINT|INT|DINT|LINT|USINT|UINT|UDINT|ULINT) #REQUIRED
  InitialValue CDATA #IMPLIED >

<!ELEMENT Subrange EMPTY>
<!ATTLIST Subrange
  LowerLimit CDATA #REQUIRED
  UpperLimit CDATA #REQUIRED >

<!ELEMENT ArrayType (Subrange)+>
<!ATTLIST ArrayType
  BaseType CDATA #REQUIRED
  InitialValues CDATA #IMPLIED >

<!ELEMENT StructuredType
(VarDeclaration|ArrayVarDeclaration|SubrangeVarDeclaration)+>

<!ELEMENT ArrayVarDeclaration (Subrange+) >
<!ATTLIST ArrayVarDeclaration
  Name ID #REQUIRED
  Type CDATA #REQUIRED
  InitialValues CDATA #IMPLIED >

<!ELEMENT SubrangeVarDeclaration (Subrange?) >
<!ATTLIST SubrangeVarDeclaration
  Name ID #REQUIRED
  Type (SINT|INT|DINT|LINT|USINT|UINT|UDINT|ULINT) #REQUIRED
  InitialValue CDATA #IMPLIED >
```

Table 2 – FBMGT DTD elements

Element attributes	Textual syntax (IEC 61131-3, Annex B)	Explanation
Request	--	An XML-encoded management request.
ID	--	A unique identifier for the Request/Response transaction.
Action	--	The requested operation to be performed as defined in Table 7
Response	--	XML-encoded management response
ID	--	Unique identifier for the Request/Response transaction
Reason	--	Reason for failure to perform a requested action (see Table 3). If absent, the action has been successfully performed
NameList	identifier {',' identifier}	A list of function block type names or data type names
FBList	fb_instance_name {',' fb_instance_name}	A list of function block instance names
FBStatus		See Figure 24 of IEC 61499-1
ByteData	--	Implementation-dependent data, typically encoded in hexadecimal format
VersionInfo	--	Currently loaded or to-be-loaded version of a function block type or data type
Organization	--	Organization supplying this library element
Date	--	Release date of this version in YYYY-MM-DD format
FB	Function block or resource instance	
Name	Identifier	Function block or resource instance name
Type	Identifier	Function block or resource type name
Connection	Event connection, data connection or adapter connection	
Source	See Note 1	
Destination	See Note 1	
VarDeclaration	Declaration of a variable	
Name	input_variable_name|output_variable_name	See Note 2
Type	data_type_name	
ArraySize	See Note 3	
InitialValue	See Note 4	
FBType	Function block type as defined in IEC 61499-1	
Name	fb_type_name	
Event	Declaration of an event interface	
Name	event_input_name | event_output_name	See Note 5
Type	event_type	
With	(input_variable_name {','input_variable_name}) | (output_variable_name {',' output_variable_name})	See Note 6
AdapterDeclaration	Declaration of a plug or socket interface of a function block type	
Name	plug_name | socket_name	See Note 7
Type	adapter_type_name	See Note 7

Table 2 – FBMGT DTD elements

Element attributes	Textual syntax (IEC 61131-3, Annex B)	Explanation
AdapterType	Declaration of an adapter interface type as defined in 2.5 of IEC 61499-1	
Name	Adapter_type_name	
DataType	Name of a data type as defined in IEC 61131-3	
Name	data_type_name	
ASN1Tag	ASN.1 tag as defined in ISO/IEC 8824	
Class	ASN.1 tag class as defined in ISO/IEC 8824	
Number	ASN.1 tag number as defined in ISO/IEC 8824	
DirectlyDerivedType	Directly derived type as defined in IEC 61131-3	
BaseType	elementary_type_name	
InitialValue	constant	
EnumeratedType	Same as NameList	A comma-separated list of enumerated values
InitialValue	identifier	If present, shall be one of the list elements
SubrangeType	Subrange type as defined in IEC 61131-3	
BaseType	integer_type_name	
InitialValue	signed_integer	
Subrange	Subrange as defined in IEC 61131-3	
LowerLimit	signed_integer	
UpperLimit	signed_integer	
ArrayType	Array type as defined in IEC 61131-3	
BaseType	non_generic_type_name	
InitialValues	array_initialization	
UpperLimit	signed_integer	
StructuredType	Structured data type as defined in IEC 61131-3	
ArrayVarDeclaration	Declaration of an array as defined in IEC 61131-3	
Name	structure_element_name	
Type	array_type_name	
InitialValues	array_initialization	
SubrangeVarDeclaration	Declaration of a subrange variable as defined in IEC 61131-3	
Name	structure_element_name	
Type	integer_type_name	
InitialValue	signed_integer	