Annex H
(informative – *proposed*)
Interoperability with IEC 61499 devices

H.1 Introduction

A programmable controller may act as a *server,* as defined in IEC 61131-5, to a *device* as defined in IEC 61499-1, acting as a *client* as defined in IEC 61131-5. These services are provided using the means defined in IEC 61131-5, and are accessed from the IEC 61499 device using instances of the *function block types* specifed in this Annex. These function block types are modeled as *communication function block types* as defined in IEC 61499-1.

The IEC 61499 client device may exist on a communication network along with the programmable controller acting as a server, or may be an implementer-specific subsystem within the “main processing unit” of the programmable controller, as illustrated in Figure 4 of IEC 61131-5:2000. In either case, the interaction between the IEC 61499 client device and the main processing unit is modeled as occurring over one or more *communication connections* as defined in IEC 61499-1, utilizing instances of the function block types defined in this Annex.

H.2 Service conventions

Except for the extensions defined in this Annex, the conventions for naming of input and output variables and events, and for describing the *services* (as defined in IEC 61499-1) provided by instances of the function block types described in this Annex, are as defined in IEC 61499-1 for the descriptions of *service interface function block types* and *communication function block types.*

For the purposes of this Annex, the `PARAMS` input of type ANY defined in IEC 61499-1 is replaced by an `ID` input of type `WSTRING`. The contents of this string specify an **implementation-dependent** representation of the path to the *variable* of interest in the server.

EXAMPLE 1 In the case where the IEC 61499 client device is in logical proximity to the IEC 61131 server, it may be sufficient to simply name the *access path* to the desired variable in the `ID` input, for instance “`CELL_1.CHARLIE`” in the example shown in Figure 19a.

EXAMPLE 2 In the case where the IEC 61499 client device is remotely connected to the IEC 61131-3 server via a communication network, it may be possible to use the `ID` input to encapsulate a Universal Resource Identifier (URI) to specify the desired access path, for instance, “`http://192.168.0.1:61131/CELL_1.CHARLIE`”.

NOTE Where supported by an implementation, the `ID` input may specify an access path to a status variable, such as the pre-defined access paths `P_PCSTATUS` and `P_PCSTATE` specified in IEC 61131-5.

Where used, the contents of the `TYPE` input of a function block type defined in this Annex specify the name of the *data type* of the data (`SD` or `RD`) being transferred. This may be the name of an elementary data type such as “`BOOL`” or a derived data type such as “`ANALOG_16_INPUT_DATAI`”.

Where used, the contents of the `TASK` input of a function block type defined in this Annex specify an **implementation-dependent** representation of the path to the *task* of interest in the server.

EXAMPLE 3 In the case where an IEC 61499 client device is in logical proximity to an IEC 61131-3 server configured as shown in Figure 19a, a path to the task named `SLOW_1` in resource `STATION_1` could be represented as “`CELL_1.STATION_1.SLOW_1`”.

Values of the `STATUS` output of the function block types defined in H.3 are as given in Table 24 of IEC 61131-5:2000.

H.3 Function block types

H.3.1 READ

An instance of the READ function block type shown graphically in Figure H.1 and textually in Table H.1 can be used by an IEC 61499 client device to read program or status variable values from an IEC 61131-3 server.

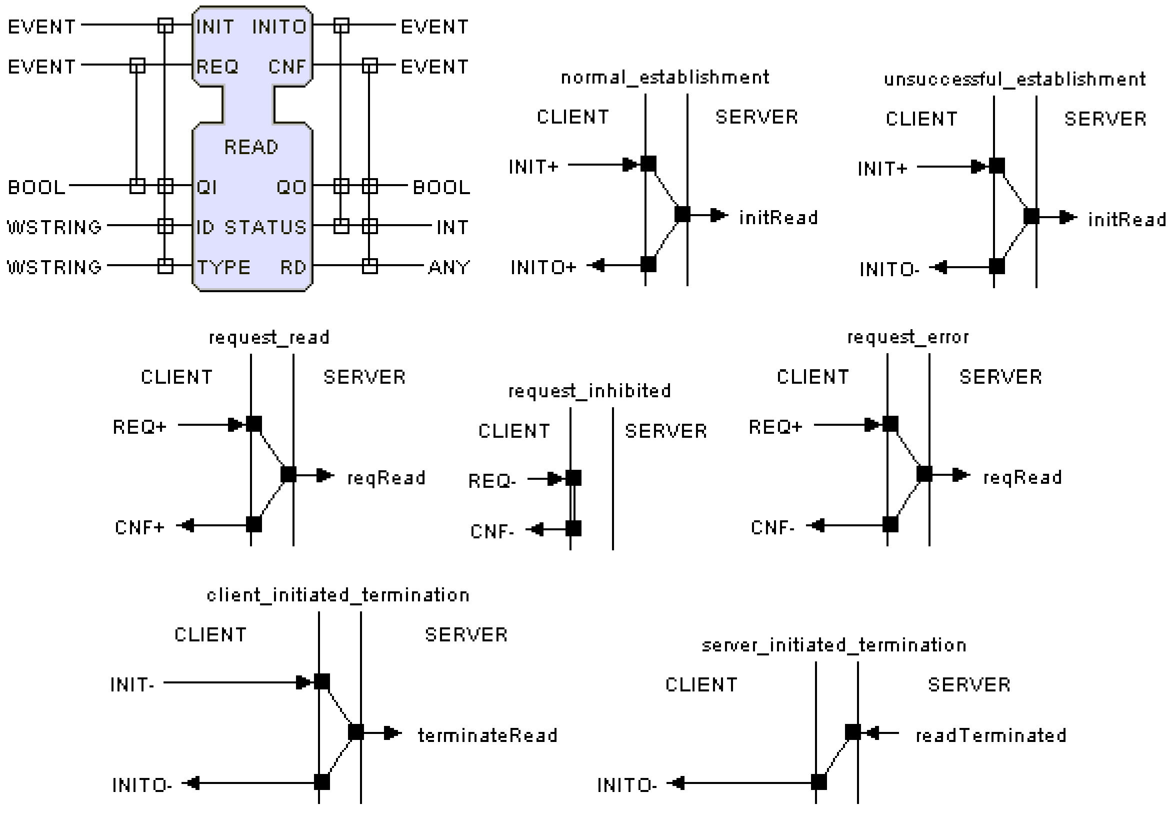


Figure H.1 – Function block type READ

Table H.1 – Source code of function block type READ

```
FUNCTION_BLOCK READ (* Read server status or program variable *)
EVENT_INPUT
    INIT WITH QI,ID,TYPE; (* Initialize/Terminate Service *)
    REQ WITH QI; (* Service Request *)
END_EVENT

EVENT_OUTPUT
    INITO WITH QO,STATUS; (* Initialize/Terminate Confirm *)
    CNF WITH QO,STATUS,RD; (* Confirmation of Requested Service *)
END_EVENT

VAR_INPUT
    QI : BOOL; (* Event Input Qualifier *)
    ID : WSTRING; (* Path to variable to be read *)
    TYPE : WSTRING; (* Data type of RD variable *)
END_VAR
```

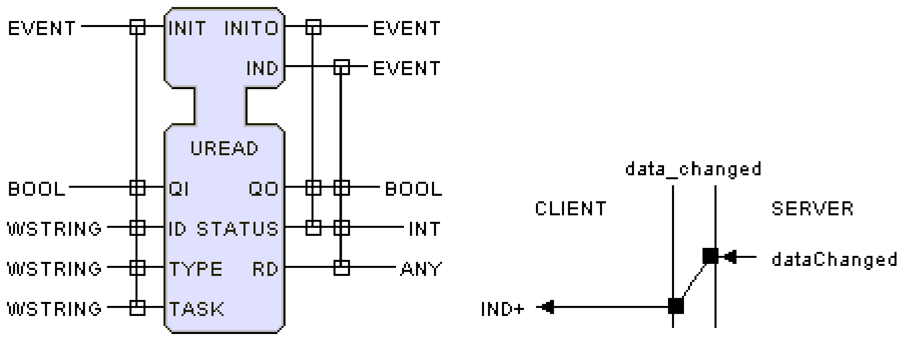
Table H.1 – Source code of function block type READ

```
VAR_OUTPUT
    QO : BOOL; (* 1=Normal operation, 0=Abnormal operation *)
    STATUS : INT;
    RD : ANY; (* Variable data from IEC 61131 device *)
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
    CLIENT.INIT+(ID,TYPE) -> SERVER.initRead(ID,TYPE) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
    CLIENT.INIT+(ID,TYPE) -> SERVER.initRead(ID,TYPE) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_read
    CLIENT.REQ+() -> SERVER.reqRead(ID) -> CLIENT.CNF+(RD);
END_SEQUENCE

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

SEQUENCE request_error
    CLIENT.REQ+() -> SERVER.reqRead(ID) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
    CLIENT.INIT-() -> SERVER.terminateRead(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
    SERVER.readTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

H.3.2 UREAD

An instance of the `UREAD` function block type shown graphically in Figure H.2 and textually in Table H.2 can be used by an IEC 61499 client device to request asynchronous notification of a change in value of a program or status variable from an IEC 61131-3 server. Notification is received via the block's `IND` event output upon completion of the execution of the specified task when a change in the value of the specified variable (with respect to its value upon initiation of task execution) is detected.

An instance of this function block type can also be used to receive notification of the completion of each execution of the specified task by leaving unspecified the `ID` and `TYPE` inputs of the block.



NOTE The graphical representation of other service sequences listed in Table H.2 is similar to Figure H.1.

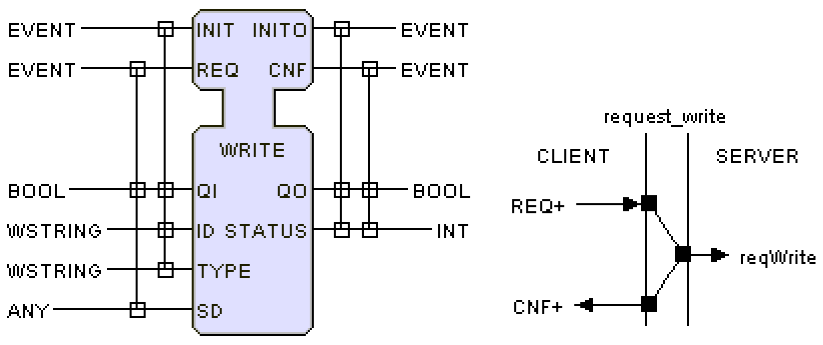
Figure H.2 – Function block type UREAD

Table H.2 – Source code of function block type UREAD

```
FUNCTION_BLOCK UREAD (* Unsolicited read of IEC 61131 program or status variable *)

EVENT_INPUT
    INIT WITH QI,ID,TASK,TYPE; (* Initialize/Terminate Service *)
END_EVENT

EVENT_OUTPUT
    INITO WITH QO,STATUS; (* Initialize/Terminate Confirm *)
    IND WITH QO,STATUS,RD; (* Indication of changed RD value *)
END_EVENT

VAR_INPUT
    QI : BOOL; (* Event Input Qualifier *)
    ID : WSTRING; (* Path to variable to be read *)
    TYPE : WSTRING; (* Data type of RD variable *)
    TASK : WSTRING; (* Path to IEC 61131 TASK triggering read on changed value *)
END_VAR

VAR_OUTPUT
    QO : BOOL; (* 1=Normal operation, 0=Abnormal operation *)
    STATUS : INT;
    RD : ANY; (* Input data from resource *)
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
    CLIENT.INIT+(ID,TYPE,TASK) -> SERVER.initURead(ID,TYPE,TASK) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
    CLIENT.INIT+(ID,TYPE,TASK) -> SERVER.initURead(ID,TYPE,TASK)
       -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE data_changed
    SERVER.dataChanged() -> CLIENT.IND+(RD);
END_SEQUENCE

SEQUENCE client_initiated_termination
    CLIENT.INIT-() -> SERVER.terminateURead() -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
    SERVER.UReadTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

H.3.3 WRITE

An instance of the `WRITE` function block type shown graphically in Figure H.3 and textually in Table H.3 can be used by an IEC 61499 client device to write variable data values to an IEC 61131-3 server.



NOTE The graphical representation of other service sequences listed in Table H.3 is similar to Figure H.1.

Figure H.3 – Function block type WRITE

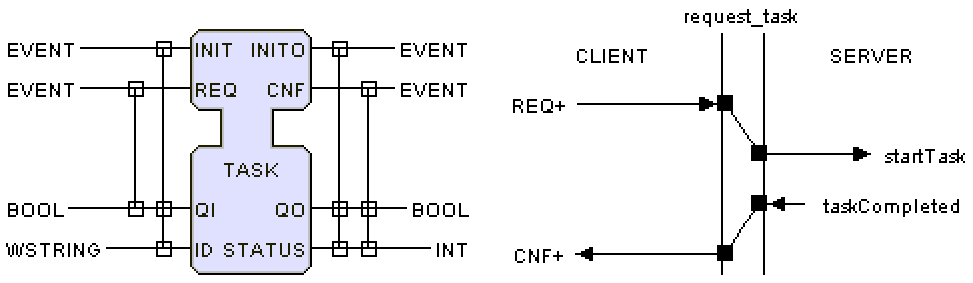
Table H.3 – Source code of function block type WRITE

```
FUNCTION_BLOCK WRITE (* Write a variable value to an IEC 61131 server *)
EVENT_INPUT
    INIT WITH QI,ID,TYPE; (* Initialize/Terminate Service *)
    REQ WITH QI,SD; (* Service Request *)
END_EVENT

EVENT_OUTPUT
    INITO WITH QO,STATUS; (* Initialize/Terminate Confirm *)
    CNF WITH QO,STATUS; (* Confirmation of Requested Service *)
END_EVENT

VAR_INPUT
    QI : BOOL; (* Event Input Qualifier *)
    ID : WSTRING; (* Path to variable to be read *)
    TYPE : WSTRING; (* Data type of SD variable *)
    SD : ANY; (* Variable value to write *)
END_VAR

VAR_OUTPUT
    QO : BOOL; (* 1=Normal operation, 0=Abnormal operation *)
    STATUS : INT;
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
    CLIENT.INIT+(ID,TYPE) -> SERVER.initWrite(ID,TYPE) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
    CLIENT.INIT+(ID,TYPE) -> SERVER.initWrite(ID,TYPE) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_write
    CLIENT.REQ+(ID,SD) -> SERVER.reqWrite(ID,SD) -> CLIENT.CNF+();
END_SEQUENCE

SEQUENCE request_inhibited
    CLIENT.REQ-(ID,SD) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
    CLIENT.REQ+(ID,SD) -> SERVER.reqWrite(ID,SD) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
    CLIENT.INIT-() -> SERVER.terminateWrite(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
    SERVER.writeTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

H.3.4 TASK

An instance of the TASK function block type shown graphically in Figure H.4 and textually in Table H.4 can be used by an IEC 61499 client device to request the execution of a task on an IEC 61131-3 server.

When an implementation supports this feature, no value is configured for either the SINGLE or INTERVAL input of the corresponding TASK block as defined in Table 50 of this Part, and execution of the corresponding task is triggered as shown in the request_task service sequence shown in Figure H.4.



NOTE The graphical representation of other service sequences listed in Table H.4 is similar to Figure H.1.

Figure H.4 – Function block type TASK

Table H.4 – Source code of function block type TASK

```
FUNCTION_BLOCK TASK (* Trigger IEC 61131 task *)
EVENT_INPUT
    INIT WITH QI,ID; (* Initialize/Terminate Service *)
    REQ WITH QI; (* Service Request *)
END_EVENT

EVENT_OUTPUT
    INITO WITH QO,STATUS; (* Initialize/Terminate Confirm *)
    CNF WITH QO,STATUS; (* Confirmation of Requested Service *)
END_EVENT

VAR_INPUT
    QI : BOOL; (* Event Input Qualifier *)
    ID : WSTRING; (* Path to task to be triggered *)
END_VAR

VAR_OUTPUT
    QO : BOOL; (* 1=Normal operation, 0=Abnormal operation *)
    STATUS : INT;
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
    CLIENT.INIT+(ID) -> SERVER.initTask(ID) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
    CLIENT.INIT+(ID,TYPE) -> SERVER.init(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_task
    CLIENT.REQ+(ID) -> SERVER.reqTask(ID) -> CLIENT.CNF+();
END_SEQUENCE

SEQUENCE request_inhibited
    CLIENT.REQ-() -> CLIENT.CNF-(STATUS);
END_SEQUENCE
```

Table H.4 – Source code of function block type TASK

```
SEQUENCE request_error
    CLIENT.REQ+(ID) -> SERVER.reqTask(ID) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
    CLIENT.INIT-() -> SERVER.terminateTask(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
    SERVER.taskTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

H.4 Compliance

The specifications given in this Annex may be referenced in **Compliance Profiles** according to the rules given in IEC 61499-4.

When a programmable controller system compliant with this Part of IEC 61131 supports interoperability with one or more of the IEC 61499 function block types defined in this Annex, it should include in its list of supported features a reference to the supported features taken from Table H.5, and should include specifications of the values for implementation specific features and parameters as defined in subclauses 8.1 and 8.2 of IEC 61131-5, respectively.

Table H.5 – IEC 61499 interoperability features

Feature No.	Description
1	READ function block type
2	UREAD function block type
3	WRITE function block type
4	TASK function block type