

CONTROLLED DISTRIBUTION ONLY IF COLOUR STAMPED

26 JUN 92

## Resource Manager


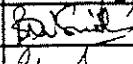
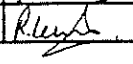
### Resource Manager Release 1 Specification

#### CONTROLLED DISTRIBUTION LIST

Brian Smith	
John Juer	
Bob Lewis	
Nigel Kingsley	

MASTER

Will be slightly revised before  
final release

APPROVAL FOR DOCUMENT REVISION	Author 	Checked	ELECTRONICALLY STORED DOCUMENT DIRECTORY PATH	
JOB TITLE	Signature	Date	pine:/users/cell/resource/docs/specs/spec1.tex	
NO SIGNATORY			ORIGINATING DEPT: ENGINEERING	
Systems Technical Director	N. Kingsley	31/5/91	CONTROLLING DEPT: ENGINEERING	
Engineering Technical Director		6/6/91	CONTROL SHEET	NO. of SHEETS 27
Chief Software Engineer		3/5/91		
AUTHOR: J W Juer			DOC. TYPE: Software Design Enhancement Specification	
CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	Resource Manager		
	EI	Resource Manager Release 1 Specification		
EUROTHERM © Copyright 1991 Eurotherm Limited		DOCUMENT NUMBER HP024105	SHT. 1	

# REVISION HISTORY

Revision	Date	Changes
0	May 13, 1991	Initial Draft
1	May 31, 1991	First issue version

## Contents

1	Scope	3
2	Related Documents	3
3	Introduction	4
4	Data exchange	6
4.1	Var Reference . . . . .	6
4.1.1	Specifying the remote data objects . . . . .	7
4.1.2	Reading the remote information . . . . .	11
4.1.3	Reading and writing remote data . . . . .	12
4.1.4	Messaging and task buffers . . . . .	13
4.1.5	Examining the state of a vref . . . . .	13
4.1.6	Timeouts and failures . . . . .	14
4.1.7	Summary of Properties . . . . .	15
5	Resources and Tasks	15
6	Networks and network setup	17
6.1	UDP . . . . .	17
6.2	EILIN . . . . .	17
6.3	Network Bridging . . . . .	18

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION 1	Resource Manager  Resource Manager Release 1 Specification	
	<b>EUROTHERM</b> © Copyright 1991 Eurotherm Limited	<b>EI</b> DOCUMENT NUMBER HP024105	SHT. 2

7	Tools	18
7.1	Task and loaders	19
7.2	CmsEars	19
7.3	CmsSpy	19
7.4	RouMsg	20
8	Debugging facilities	20
8.1	The Resource Debugger	20
8.2	Error monitoring blocks	24
9	Features for PO 2.5	27
10	Features for PO 2.6	27

## 1 Scope

This specification describes the features to be provided by the first release of the Resource Manager. Initially this will affect the Production Orchestrator, later on the PC3000. For the purposes of this document, however, the Resource Manager will be treated as an enhancement to the Production Orchestrator only, and use of the Resource Manager in the PC3000 will be a separate project.

Two releases of the Resource Manager for the Production Orchestrator will be made, and these are both described here. These releases will have to be co-ordinated with the first release of EILIN, (see document [1] )

## 2 Related Documents

- [1] HP024106 EILIN
- [2] HP024081C300 PC3000 Multi-Tasking system
- [3] IEC65A/WG6 Programmable Controllers Programming Languages

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager	
		Resource Manager Release 1 Specification	
<b>EUROTHERM</b> © Copyright 1991 Eurotherm Limited	<b>EI</b>	DOCUMENT NUMBER	SHT.
		HP024105	3

### 3 Introduction

A Resource is defined in reference [3] section 1.2.1 as a "signal processing function", its "man-machine interface" and "sensor-actuator" interface"; in the context of this document two Eurotherm products can be considered as Resources — the PC3000 and the Production Orchestrator. In more general terms we consider a Resource to be a node on a network (or networks) such as ethernet or EILIN that executes a set of Structured Text tasks as defined in reference [3] which perform control and display of process or production data (i.e the functions described in 1.2.1 of IEC65 [3]).

In IEC65 a Resource is actually a ST language construct that encapsulates the tasks and programs running in a physical Resource; our implementation of ST is extended to support this construct as part of the Resource Manager project. (The mechanism in reference [3] for data exchange between Resources is however not supported by the Resource Manager which has a more general purpose and flexible mechanism, described in this document.)

The Resource Manager is the code and supporting data built from the definition of an application in ST that allows that application running on one Resource in a network to import and export data from any other nodes on the network in a reasonably simple and transparent way. If the node supports a deterministic network such as EILIN and a deterministic task execution model (such as in the PC3000) this will allow Structured Text applications to perform distributed control.

The Resource Manager provides

- A Structured Text database for a Resource, built using the ST compiler and a Resource loader
- Methods for interrogating the database from any ST task, and interactively using a debugger
- Methods for monitoring the data interchange between tasks, and hence for tuning applications

In the rest of this document the word "Resource" is used to denote the Resource Manager code, the Resource Manager database, the physical node or Resource the code and database exists in and the ST code actually being executed. The context should make it clear which is meant.

Figure 1 shows an example of some Resources and tasks. The boxes labelled "Node 1" and "Node 2" represent two hardware Resources (for example 386 computers) connected by a network. Node 1 contains a Resource with two tasks, "task1" and "task2". Node 2 is a Resource running only "task3". Task1 is in communication with task2 using the local Resource database, task2 is communicating with task3 using both Resource databases and the network.

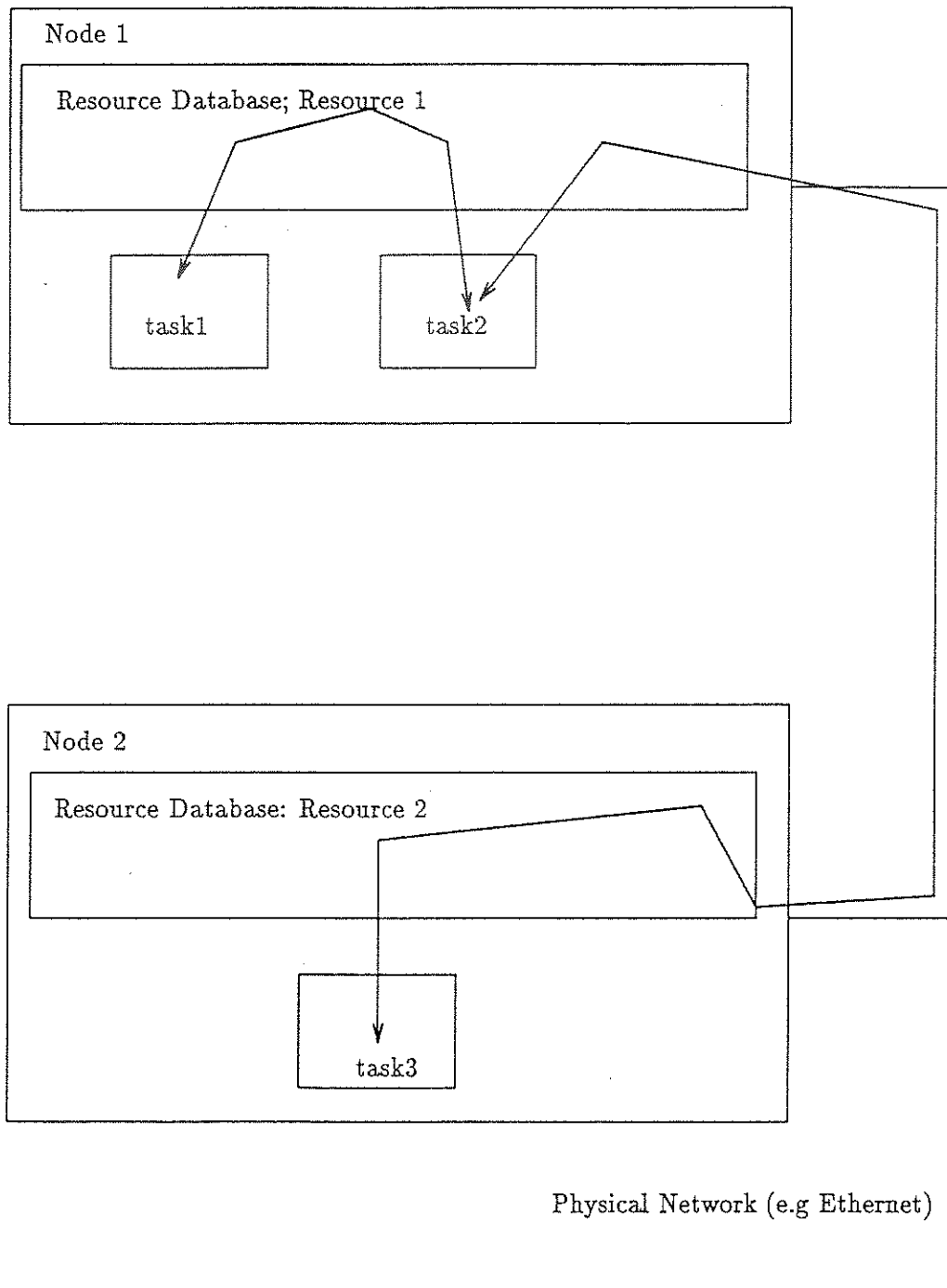
Remote data can be grouped (or blocked) as required by the local application. Remote data appears in a local function block or variable as if it were local. Remote points from disparate remote ST objects can be treated as one local object, provided they all are owned by one remote Structured Text task (and hence one remote Resource). So, for example a set of remote setpoints can be written or read in one communication transaction.

The Resource Manager also allows for a limited degree of interactive Structured Text debugging (across a network if required); ST data may be displayed and printed and task execution suspended on entry or exit from a block, or the whole task.

As suggested by figure 1 the Resource Manager works using messages; tasks receive messages in a message input queue, and process messages at the beginning of their task execution cycle. A shared Resource database held in RAM allows access to the ST data owned by a task (i.e the data of the blocks the task is executing). Tasks do not know whether or not they are sending local messages (to a task in the same Resource) or remote (to a task in another Resource); this is handled by the Communication Messaging Services (CMS). A special built in task called the Router deals with incoming and outgoing messages to remote Resources. It can also

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager	
		Resource Manager Release 1 Specification	
<b>EUROTHERM</b> © Copyright 1991 Eurotherm Limited	<b>EI</b>	DOCUMENT NUMBER	SHT.
		HP024105	4

Figure 1: Resources and Tasks on a network



CONTROLLED DISTRIBUTION COPY  
ONLY IF COLOUR STAMPED ON  
CONTROL SHEET.

DOCUMENT  
REVISION

1

Resource Manager

Resource Manager Release 1 Specification

**EUROTHERM**

© Copyright 1991 Eurotherm Limited

**EI**

DOCUMENT NUMBER

HP024105

SHT.

5

pass messages from one remote Resource to another remote Resource allowing routing of messages between EILIN and ethernet for example. The Resource Manager code allows for graceful failure and recovery when messages are not delivered.

This design means that any data exchanged is automatically "coherent" as defined in [3] section 2.7.2 (i.e all data received by one task from another in one message is from the same cycle of the other task. This is very important for control applications. It also means that the design allows for deterministic execution of tasks — tasks are under no obligation to reply to a message if they have not the time, they can merely ignore it and the system will recover.

The design is also general purpose; it supports multiple processors accessing a common shared memory Resource database and single processors on boards connected by a bus on one Resource. In the latter case the Resource database is duplicated in the memory of each processor card, and the Router task on each board deals with messaging between boards.

## 4 Data exchange

The Structured Text language has been extended to allow import of remote data. Whereas in [3] Resources exchange information through fixed predefined communication blocks this extension allows for

- Interrogation of a Resource to query what data is available and validate any data exchange
- Grouping of remote data to provide the local view as is required by the local application

This section describes the mechanisms that allow this. (Note later releases of the Resource will allow CDL attributes — in particular the attribute "addressable" will allow some data items to be inaccessible).

### 4.1 Var Reference

Every Function Block or Program can have an additional declaration section, the reference section. All data specified in this section is remote data, where remote means that it has its defining definition somewhere other than in the current block, though it may well be in the same Resource or indeed task. The reference section is denoted by the keyword REFERENCE appearing after the keyword VAR. Note therefore that references are internal to the block they are defined in.

For example

```
PROGRAM ex1
VAR
  writeflag: BOOL;
  id: STRING;
END_VAR
VAR REFERENCE
  remflag: BOOL;
  remid: STRING;
END_VAR
```

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager	
EUROTHERM  © Copyright 1991 Eurotherm Limited	EI	Resource Manager Release 1 Specification	
		DOCUMENT NUMBER HP024105	SHT.  6

has a reference section which contains two remote objects remflag and remid.

Any object may be declared to be a reference, from simple variables of any type, to arrays of any type, to function blocks.

References have a set of "properties" which are predefined built in variables. Properties may be assigned or read or both. Properties cannot be wired to. Properties are used to control and monitor the reading and writing of data via the reference.

The first stage in accessing remote data is specifying where it is and then matching it to the local data. If this operation is successful then the remote data may be read and/or written.

#### 4.1.1 Specifying the remote data objects

A var reference (hereafter known as vref) has an associated string, the ref string. This is set by assigning it from within the ST program, for example

```
remflag~ref := 'Res1:pid1.in';
```

The ~ tells the ST compiler that the next name is a "property" of a vref object. Property names are predefined, and the ref property is the reference string. Any ST string or ST string expression may be assigned to it.

A property called curref can be used to read the last set reference string.

**Simple types** For a simple vref, that is one which is a simple built in ST type (e.g DINT, BOOL, LREAL) the reference string must specify the full hierarchic path to the object, prefixed by an optional Resource name and ":". The syntax is

```
simple_ref_string ::= [ resource_name ] ':' name { '.' name }
```

(using the usual BNF notation where [] means an option, and {} means 0 or more of the enclosed). name is any valid ST name.

The resource\_name is the name of a remote Resource. If omitted the reference is to something in the local Resource. The list of names separated by . is the full path to the remote object. So in the above example Res1 is the name of the remote Resource, pid1 is a block in the remote Resource which contains a variable in.

When a reference string is assigned the Resource will query the specified remote Resource for information about the specified object. The information returned is

- The type of the remote object (DINT, LREAL etc.)
- The mode of the remote object (input, output etc.)
- The size of the remote object, which will be 1 for simple types and the total number of elements for an array type.
- A fast address for the remote object
- The task that owns the remote object

In order for reads and writes to be performed, the type and size must match the type, size and mode (i.e input, output, internal, in-out) of the local vref. In fact for simple types (i.e vrefs that are not blocks) the mode of the remote object must be internal, since the vref is itself internal.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager	
		Resource Manager Release 1 Specification	
<b>EUROTHERM</b> © Copyright 1991 Eurotherm Limited	<b>EI</b>	DOCUMENT NUMBER	SHT.
		HP024105	7

**Arrays** For arrays of simple variables, as well as the type and mode matching, the total number of elements in the remote object must match the number in the local object. So, for example a remote 2 by 10 array would match a local 10 by 2 array. In general a remote array with 6 dimensions  $i_1, i_2, i_3, i_4, i_5, i_6$  matches a local one with dimensions  $j_1, j_2, j_3, j_4, j_5, j_6$  provided  $i_1 * i_2 * i_3 * i_4 * i_5 * i_6 = j_1 * j_2 * j_3 * j_4 * j_5 * j_6$ . Local data at position  $x_1, x_2, x_3, x_4, x_5, x_6$  would be the remote data at position  $y_1, y_2, y_3, y_4, y_5, y_6$  if the same position has been specified when the array is "flattened" into a one-dimensional array. Since  $i_6$  and  $j_6$  are the fastest varying dimensions this means

$$x_6 + i_6 * (x_5 - 1 + i_5 * (x_4 - 1 + i_4 * (x_3 - 1 + i_3 * (x_2 - 1 + i_2 * (x_1 - 1)))))) = y_6 + j_6 * (y_5 - 1 + j_5 * (y_4 - 1 + j_4 * (y_3 - 1 + j_3 * (y_2 - 1 + j_2 * (y_1 - 1))))))$$

With the constraints for the array indices (e.g  $0 < x_6 < i_6$ ) this gives a unique mapping between one array and another.

It is also possible to match to single elements of an array, or to the whole of a sub-array, in exactly the same way as the Structured Text compiler allows array assignment.

For example given a remote array declared as

```
array: ARRAY[1..10,1..10] OF DINT;
```

and the local declaration

VAR REFERENCE

```
matchall: ARRAY[1..10,1..10] OF DINT;
matchpart: ARRAY [1..10] OF DINT;
matchele: DINT;
```

then the following would be valid.

```
matchall~ref := 'Remote:prog.array'
matchpart~ref := 'Remote:prog.array[2]'
matchele~ref := 'Remote:prog.array[1,2]'
```

So it is possible in the reference string to index into remote arrays, and have a successful match provided the dimension of the local object matches.

Note that a local array object can only have a single reference string (not an array of them).

CONTROLLED DISTRIBUTION COPY  
ONLY IF COLOUR STAMPED ON  
CONTROL SHEET.

**EUROTHERM**

© Copyright 1991 Eurotherm Limited

DOCUMENT  
REVISION

1

**EI**

Resource Manager

Resource Manager Release 1 Specification

DOCUMENT NUMBER

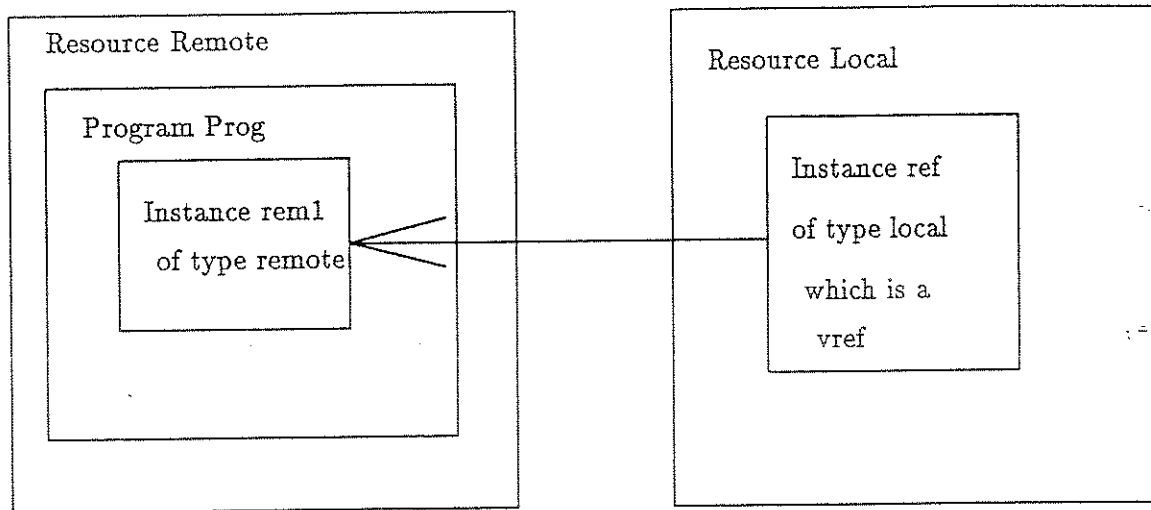
HP024105

SHT.

8



Figure 2: A reference to a remote block



**Function Blocks** A vref can also be a function block. The reference string then specifies one or more remote data objects that are matched to the inputs, outputs and in-outs of the local object. In the simplest case where only one remote object is specified, the remote object must be a block with parameters which match the local object's parameters in name, mode, type and size except that a remote internal may match a local input, output, in-out or internal. In other words each parameter of the remote object is individually matched to the local object's parameters by name as if it were a simple type. (The remote block may have extra parameters that are not matched). The local block is then an image of (possibly part of) the remote block's data.

The diagram in figure 2 should help understanding of the example shown below. Given a remote block such as

```

FUNCTION_BLOCK remote
VAR_INPUT
    in1:LREAL;
    in2: ARRAY[1..10] OF DINT;
END_VAR
VAR_OUTPUT
    out1: BOOL;
    ignored: BOOL;
END_VAR
  
```

which was instantiated in a Resource called Remote in a program called prog as block instance rem1, and a local block definition of the form

```

FUNCTION_BLOCK local
VAR_INPUT
    in1:LREAL;
    in2: ARRAY[1..10] OF DINT;
END_VAR
  
```

CONTROLLED DISTRIBUTION COPY  
ONLY IF COLOUR STAMPED ON  
CONTROL SHEET.

DOCUMENT  
REVISION

1

Resource Manager

Resource Manager Release 1 Specification

**EUROTHERM**

© Copyright 1991 Eurotherm Limited

**EI**

DOCUMENT NUMBER

HP024105

SHT.

9

```
VAR_OUTPUT
  out1: BOOL;
END_VAR
```

then the following vref

```
VAR REFERENCE
  ref:local;
END_VAR
```

```
ref^ref := 'Remote:prog.rem1';
```

would match the in1, in2 and out1 parameters, and would enable data to be exchanged via those parameters.

Of course by instantiating a local instance of remote all of its visible parameters may be matched.

It is also possible to specify a list of remote objects that are to be matched to a local block, by using a special syntax in the reference string. Fully hierarchic names can be put in a comma separated lists, or as a shorthand '{' and '}' are used to bracket comma separated lists of names which are then all taken to be relative to the previous hierarchic name. For example the string

```
a{b,x.e,f{j,k,l{m,n}}}
```

expands to the names

```
a.b, a.x.e, a.f.j, a.f.k, a.f.l.m, a.f.l.n
```

A local parameter must be assigned to each name in the resulting expanded list, for example

```
a{ loc1 := b, c { loc2 := d, loc3 := e}}
```

means that the local parameter loc1 is matched to a.b, loc2 to a.c.d and loc3 to a.c.e.

The full syntax of the reference string is

```
ref_string ::= simple_ref_string | complex_ref_string
simple_ref_string ::= [ resource_name ] ':' name { '.' name }
complex_ref_string ::= [ resource_name ] ':' [ primary_name ]
                      '{' alternate_names_list '}'
alternate_names_list ::= alternate_names { , alternate_names_list }
alternate_names ::= [ hierarchic_name ] '{' alternate_names_list '}' |
                    final_name
final_name ::= local_parameter_name ':' hierarchic_name
```

CONTROLLED DISTRIBUTION COPY  
ONLY IF COLOUR STAMPED ON  
CONTROL SHEET.

DOCUMENT  
REVISION  
1

Resource Manager

Resource Manager Release 1 Specification

**EUROTHERM**

© Copyright 1991 Eurotherm Limited

**EI**

DOCUMENT NUMBER  
HP024105

SHT.

10

```

primary_name ::= name { '.' name }

hierarchic_name ::= name { '.' name }

local_parameter_name ::= name

resource_name ::= name

```

The `primary_name` is the name relative to which all the following list of names is specified. If absent the following list of names is taken relative to the remote Resource as a whole, (i.e the list of names must contain the full path to the object). The { and } notation brackets a comma separated list of hierarchic names. Any name may itself contain a list of sub-objects using the { and } notation. At the bottom level the `local_parameter_name` specifies the parameter of the local vref that is to be matched to the remote name. Any parameters of the local vref that are not explicitly assigned remote objects are matched to parameters of the same name in the `primary_name`; if the latter is absent this is an error.

Duplicate `local_parameter_names` are not allowed, but duplicate remote names are, so it is possible to match one remote variable to two or more local variables.

For example

```

VAR REFERENCE
  RP : RemPID;
END_VAR
  RemPID^ref := 'R1:a{ Sp := b,c{ Pv := d, Op := e}'

```

means that `R1:a.b` must match `RP.Sp`, `R1:a.c.d` must match `RP.Pv` and `R1:a.c.e` must match `RP.Op`. If `RP` has an extra parameter `X` then it is matched to `R1.a.X`.

#### 4.1.2 Reading the remote information

When a reference string is assigned to a vref, the Resource will work out the names of the remote objects it needs to match to the local, and send (in one message) these to the remote Resource. Various errors may then occur. These may be found by examining the vref's status property, for example

```

IF ref^status > 1 THEN
(* an error of some sort *)
END_IF

```

The errors associated with matching are

- A syntax or other error in the reference string
- The remote Resource is not reachable.
- One or more of the remote objects do not exist.
- The remote objects are owned by more than one task, and so cannot be made into one reference.
- The remote objects do not match the local ones according to the rules specified above.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager  Resource Manager Release 1 Specification	
<b>EUROTHERM</b> © Copyright 1991 Eurotherm Limited	<b>EI</b>	DOCUMENT NUMBER HP024105	SHT.  11

### 4.1.3 Reading and writing remote data

Once a reference has been matched to the remote objects, data may be read. The scan property sets the scan rate in milli seconds for the remote data. For example

```
ref^scan := t#10;
```

sets the scan rate for the reference `ref` to be once every 10 milliseconds. This means that every 10 milliseconds the Resource will send a read message to the remote Resource specified in the reference string to read all of the remote data. When the reply comes back the local image of the remote data is updated. A scan rate of zero means no reads are performed.

If, however, no reply is received by the time the next scan is due, no timeout occurs, and no message is resent. (Timeouts are handled separately with a separate global value). Thus specifying a very fast scan time means that the data will be read as fast as possible, being limited by the rate at which the remote task responds and the local task sends messages.

A scan is performed on each task cycle where 'Time now  $\geq$  Time of last scan + Scan time' ( a scan time of  $0 = \infty$  ). Therefore to perform a single-shot read, the scan time should be changed from 0 to a value  $\leq$  the task cycle time and then reset to 0 on the next task cycle.

A property called `currscan` can be used to read the last set scan rate.

For function blocks all data is read (i.e all matched outputs, inputs and in-outs) and placed in the local `vref`.

Accessing the data of the `vref` from ST always returns the last data read.

A write to the remote object is triggered by either assigning to it (if it is a simple variable) or calling it (if it is a block) passing it input parameters as usual. A write message is generated, unless a write message is already outstanding. In this case a flag is set to indicate that another write is required when the previous write is acknowledged. In this way the latest local value is always written to the remote object. Note that writes can only occur at the rate at which the remote task acknowledges them.

If the `vref` is a simple variable or array all the local data is sent (even if only part of an array has been written).

If the `vref` is a function block the inputs and in-outs of the block are sent. Note that the in-outs are not then read back, so the value assigned to the local in-out will be the value written and will not reflect any modifications made by the remote block.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION	Resource Manager	
	1	Resource Manager Release 1 Specification	
<b>EUROTHERM</b> © Copyright 1991 Eurotherm Limited	<b>EI</b>	DOCUMENT NUMBER	SHT.
		HP024105	12

#### 4.1.4 Messaging and task buffers

Tasks in a Resource communicate by exchanging messages. Each task has a configurable number of buffers of different sizes for messaging. Each task has a queue (a linked list) of free buffers, and of incoming messages. When a message is sent the smallest available buffer that will contain a message is chosen to send it. The message is delivered by linking the message buffer from the sending task into the input list for the remote task. When the remote task reads the message the buffer is freed and put back on the free list of the buffer owner (the sending task).

Messages to remote Resources, however, are routed to a "router" task. This task is responsible for finding the route to the remote Resource.

In fact the above is an over-simplification. One node on the network can be single Resource running tasks on more than one processor in a card. If the node does not support a large enough shared memory region between the processors to hold the Resource database then the database is duplicated on each card and router tasks on each processor route messages between the cards as well as to whatever network medium any card supports.

A task may run out of buffers if it is trying to send too many messages, or if the receiving task is not freeing buffers because it is stopped, dead or in error. The configurer should try to allocate Resources so that this does not happen. Various tools are available for determining the number of buffers used by tasks and error conditions. (See section 6 and 8)

All messages apart from a read template are sent to a particular task on a particular Resource. A read template message is just sent to any task on the remote Resource, since any task can interrogate the Resource database. Messages not routed to a particular task (but only to a particular Resource) are handled by the first task to receive the message, which will be the router for an inter-resource message.

Reads and writes from a task to itself (i.e. when there is a vref to objects in the same task as the vref) are handled directly without messaging. Similarly a read template request that specifies no remote Resource is handled immediately by the task issuing it.

#### 4.1.5 Examining the state of a vref

The status property of type DINT is available to determine the current state of a vref. The status property has the values and meaning shown in the following table —

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager	
		Resource Manager Release 1 Specification	
<b>EUROTHERM</b> © Copyright 1991 Eurotherm Limited	<b>EI</b>	DOCUMENT NUMBER	SHT.
		HP024105	13

State	Value	Meaning
OK	0	Last operation succeeded
InProgress	1	Read, write or read template in progress
ParseFail	2	A reference string had the wrong syntax
ResolveFail	3	Local names in the reference string did not match to the local object, or were duplicated
NoResources	4	The task has no buffers left to send messages with, or the expanded reference string is too long
TemplateMismatch	5	The read template did not match the local one
Unreachable	6	The router was not reachable
BadStatus	7	Either a read template specified non-existent objects, or read failed to read the data at the remote end (though the message arrived), or a write failed to write (for example if some block outputs were being written)
NonUniqueOwner	8	In a read template the remote objects belong to more than one remote task
SystemError	9	This should not be seen, if seen there is a internal error in the Resource

#### 4.1.6 Timeouts and failures

All read, write and read-template operations have built in timeouts, controlled by environment variables for cell 2.5. Later on for cell 2.6 task inputs will be provided for this.

On a timeout the Resource will automatically try to re-read the template of the remote objects again. This is to ensure consistency if the timeout was because a remote Resource was reloaded.

In addition each message contains a checksum for the remote Resource. This is stored in the local vref, and if a read or write returns with a different checksum to the local one then the remote Resource has been reloaded between the read or write. Again the remote template will be re-read.

The automatic re-read of templates in these circumstances means that a timeout is not visible to the user via the status property, so two other properties are available to examine the success of read or write messages. These are `readstatus` and `writestatus` ST DINTs. These both have the following states

State	Value	Meaning
OK	0	Last operation succeeded
InProgress	1	Read, write in progress
Failed	2	The last read or write failed
Undefined	3	No read or write issued with this ref string

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager	
		Resource Manager Release 1 Specification	
<b>EUROTHERM</b>  © Copyright 1991 Eurotherm Limited	<b>EI</b>	DOCUMENT NUMBER	SHT.
		HP024105	14

Reads and writes are essentially asynchronous. To simulate synchronous operations a Sequential Function Chart may be used, which tests the read/write status in a transition to determine when an operation completed.

A synchronous write may be performed by writing the remote data in a step and transitioning out of the step when the vref has OK writestatus.

A synchronous read may be performed by setting the scan property from 0 to a positive large value in a step (so that only one read will be done), transitioning out of the step when readstatus is OK, and setting the scan property to zero.

Normally, once a template has been matched successfully, there should not be communications errors. Function blocks will be available whose outputs will give information on errors and the state of the various communications interfaces, (see section 8.1).

#### 4.1.7 Summary of Properties

The following table summaries the properties that a vref has.

Property	ST Type	Mode	Meaning
ref	STRING	Input	Specifies the object(s) referred to (section 4.1.1)
status	DINT	Output	Monitor any errors using a vref (section 4.1.5)
writestatus	DINT	Output	Monitor success or failure of the last write operation (section 4.1.5)
readstatus	DINT	Output	Monitor success or failure of the last read operation (section 4.1.5)
curref	STRING	Output	The current reference string (section 4.1.1)
scan	TIME	Input	Used to set the scan rate (section 4.1.3)
currsan	TIME	Output	The current scan rate (section 4.1.3)

## 5 Resources and Tasks

The Structured Text language has been extended according to the IEC65 specification for Resource syntax. A program declaration is no longer the top level object in a Production Orchestrator configuration. The following syntax is used to specify a Resource name, and a set of tasks and programs. (Items in ' ' are literal strings)

```
resource ::= 'RESOURCE' resource_name 'ON' resource_type_name
           task_configuration_list ';'
           block_configuration_list ';'
           'END_RESOURCE'
```

```
task_configuration_list ::= task_configuration { ';' task_configuration }
```

```
task_configuration ::= 'TASK' task_name [ task_inputs ]
```

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager	
		Resource Manager Release 1 Specification	
<b>EUROTHERM</b> © Copyright 1991 Eurotherm Limited	<b>EI</b>	DOCUMENT NUMBER HP024105	SHT.  15

```

[ 'ON' processor_name ]

task_inputs ::= '(' task_input, { task_input } ')'

task_input ::= task_in ':' constant_value

block_configuration_list ::=
    block_configuration { ';' block_configuration }

block_configuration ::= 'PROGRAM'
    program_instance_name [ input_assignments ] 'WITH'
    task_name ':' program_type_name

input_assignments ::= '(' input_assignment { , input_assignment } ')'

input_assignment ::= program_input ':' constant_value

```

where

- resource\_name, task\_name, program\_instance\_name are valid ST identifiers
- program\_input is an input to the previously compiled program of type program\_type\_name
- task\_in is one of the predefined task inputs, which depends on the processor\_name and resource\_type\_name
- constant\_value is a valid ST constant, and matches the type of the program\_input or task\_in
- task\_name in block\_configuration is the name of a previously declared task
- processor\_name is an implementation dependent string denoting which processor the task will run on for a multi-processor Resource
- resource\_type\_name is an implementation dependent string denoting which target hardware the Resource will run on
- 'WITH' task\_name specifies the task that will execute the program when it is activated.

(see sections 9 and 10 for details of supported processor\_name and resource\_type\_name and corresponding task\_inputs)

Note that inter-program wiring will be supported in future releases.

A Resource declaration is compiled by the ST compiler as usual. The Resource database is then loaded. At load time each task is assigned a unique number, or task id. Loading the Resource database and running it depends on the target hardware. For the first release of the Resource manager this is described in sections 9 and 10.

CONTROLLED DISTRIBUTION COPY  
ONLY IF COLOUR STAMPED ON  
CONTROL SHEET.

DOCUMENT  
REVISION

1

Resource Manager

Resource Manager Release 1 Specification

**EUROTHERM**

© Copyright 1991 Eurotherm Limited

**EI**

DOCUMENT NUMBER

HP024105

SHT.

16



## 6 Networks and network setup

Resources may be connected together by 2 networks, either UDP over ethernet or EILIN. Production Orchestrators may support either UDP or UDP and EILIN, if a Computrol card is fitted. (UDP is a protocol commonly found on Unix that runs on ethernet.)

Where a Resource runs on more than one (closely bound) processor, there will be a communications medium between the different processors. This medium (or network) is closed and only connects the routers of that Resource. Initially the only instance of this will be the DMA interface between a 386 unix host and a Computrol card — this can be seen as an intra-resource network of 2 nodes.

Each network is independent of the other, indeed the architecture allows many other networks to be added later. Each Resource has one or more routers in it. Each Resource could potentially be a node on many networks. For every network connected to a Resource there will be a router task in the Resource. There may be additional routers for internal routing between local processors.

### 6.1 UDP

The UDP network is a connectionless peer-to-peer network, where a peer is any unix router supporting UDP. In addition there must exist somewhere on the ethernet a single copy of the UDP name server. The server exists to retain the Internet addresses of all routers and to broadcast all changes to all routers.

To start up the UDP name server

```
udp_server ['-p' port_number ] ['-f' file_name]
```

The optional file name may be used to specify a different file for holding the last known addresses of Resources, default is 'peers.udp'.

When the unix router is started the following switches must be supplied

```
'-m' 'UDP' server_hostname [port_number]
```

The default port number is 6000, for both udp server and router.

### 6.2 EILIN

The EILIN network is a peer-to-peer network, where a peer ( or node ) is any router supporting EILIN. Each Resource sets up a connection to all Resources that it communicates with.

When the Computrol router is loaded the following switches must be supplied

```
'-m' 'EILIN' node_number
```

The EILIN node number must be unique within the EILIN network.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager	
		Resource Manager Release 1 Specification	
<b>EUROTHERM</b>  © Copyright 1991 Eurotherm Limited	<b>EI</b>	DOCUMENT NUMBER	SHT.
		HP024105	17

## 6.3 Network Bridging

With more than one network medium (e.g ethernet and EILIN) it is possible to have a network of Resources where there is no common network medium between certain pairs of Resources, (i.e one could be only on ethernet and one on EILIN). In these circumstances it is necessary to bridge between the different network media. This is achieved at a Resource supporting both media (i.e on EILIN and on ethernet).

It is necessary to inform the originating router of another Resource's router which is capable of acting for the originating router to reach the destination Resource. This router is said to act as a proxy for the originating router. So, in order to set up network bridges it is necessary to set up some router(s) to use other routers as proxies. This is achieved by sending a router message to the originating routers, these messages may originate from anywhere in the network.

Each router has a list of Resources where another router is capable of acting as a proxy for each Resource not directly reachable on the media supported by the originating Resource. All routers will always act as proxy (route on) any message not for itself if at all possible.

In order to send these router messages the tool 'RouMsg' should be used, this tool sends router messages to specified destination routers and prints out their replies.

For example if Resource Res1 supports UDP, Res2 supports EILIN and Res3 supports UDP and EILIN. Then Res1 wants Res3 to act as proxy for Res2 and Res2 want Res3 to act as proxy for Res1.

The 'RouMsg' script to set up this example would be

```
Res1:Router ReqSetProxy Res2 Res3
Res2:Router ReqSetProxy Res1 Res3
```

In this example this script could be issued from either 'Res1' or 'Res3', or if the commands were issued in the opposite order then from either 'Res2' or 'Res3'. A proxy message cannot be sent to a Resource unless a path to that Resource already exists.

## 7 Tools

The Resource tools may be used to monitor the messaging of a Resource, and for unloading tasks and Resources.

The tools all run under unix.

CONTROLLED DISTRIBUTION COPY  
ONLY IF COLOUR STAMPED ON  
CONTROL SHEET.

DOCUMENT  
REVISION

1

Resource Manager

Resource Manager Release 1 Specification

**EUROTHERM**

**EI**

DOCUMENT NUMBER

HP024105

SHT.

18

© Copyright 1991 Eurotherm Limited

## 7.1 Task and loaders

On unix the output of the build of a Resource is two programs, one called 'task' and one called 'loader'. The loader will load the Resource definition into unix shared memory. The task program will run a specified ST task. The command are run as

```
loader [ -s <database size in bytes> ]
task <task name> -i <buffer spec>
```

where `buffer spec` is a string that may be used to specify the message buffer distribution for the task. It has the format

```
buffer_spec ::= number_of_buffers ':' size { buffer_spec }
```

for example

```
"10:1024 10:2048"
```

specifies 10 buffers of size 1024 bytes and 10 of size 2048 bytes.

In cell 2.6 this way of specifying task buffers will be replaced by using a task function block.

The unload program unloads a task from a Resource, or unloads a whole Resource. A task that crashes must be unloaded before it can be run again (or the whole Resource must be reloaded). Unload is run as

```
unload [ resource | task <task name> ]
```

Note an unloaded task that is run again will reuse its old buffers, and ignore any `buffer spec` supplied.

## 7.2 CmsEars

The Resource messaging system is known as CMS (Communication Messaging Services). Various utilities are provided for monitoring the state of messages and task buffers.

The CmsEars tools dynamically monitors the state of the CMS for a Resource. The tool reports summary information on the usage of buffers, statistics as well as static information about loaded tasks.

## 7.3 CmsSpy

The CmsSpy tool allows detailed inspection of the CMS data for a particular task. CmsSpy is primarily designed as a post-mortem tool, if used on a currently executing task it does not guarantee the consistency of the information. CmsSpy allows individual task buffers to be inspected.

CmsSpy has optional switches which may be applied on startup, where they change defaults or to an individual command line in which case they only affect the output from that command. These are

- '-b' print all of the buffer
- '-f' <format> print each byte in <format>, default
- '-z' count up, but do not print trailing zeros in a buffer

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager	
		Resource Manager Release 1 Specification	
<b>EUROTHERM</b>  © Copyright 1991 Eurotherm Limited	<b>EI</b>	DOCUMENT NUMBER	SHT.
		HP024105	19

## 7.4 RouMsg

The RouMsg tool allows router messages to be sent to a Resource router and the results printed out. Router messages are ONLY understood by routers. RouMsg reads single lines from stdin until EOF and executes each line. Each line is of the form

destination message\_type [ args for message\_type ]

The full set of message types and optional arguments are

**ReqListMedia** List all media supported by this Resource.

**ReqListResources** List all Resources reachable from this Resource, and the media on which they are reachable.

**ReqListProxies** List all proxies known at this Resource.

**ReqSetProxy** <Resource> <Proxy> Use <Proxy> as proxy for the resource <Resource>

**ReqCmsStatus** List the buffer distribution of the router.

**ReqMediumStatus** <Medium> Report the status of this medium and the number of messages both sent and received on this medium.

RouMsg must send messages as a Resource task, it therefore uses a task slot, but no task slot is allocated for it; it will not be a permanent feature of the Resource as its primary use is to set up proxies on a Resource that needs them. Therefore RouMsg uses a task slot currently not being used, which cannot be the router slot.

RouMsg accepts the following switches

- '-e', Echo all commands, default is no echo.
- '-i <InitData>', CMS buffer distribution.
- '-t <TaskName>', name of task whose slot to use, default 'DefaultTask'.
- '-w <Seconds>', time out an response from router, default 10 seconds.

## 8 Debugging facilities

### 8.1 The Resource Debugger

The debugger is a task that can be run on any Production Orchestrator in the network. It works by connecting to tasks on other Resources (or on the local Production Orchestrator) and sending them debug messages, to which they respond. It is run as a stand-alone program in an xterm or terminal of a unix machine running the Resource.

There are two levels of debugging, "user level" and "resource level". The latter is used for debugging the Resource Manager code and as such is outside the scope of this document.

Commands available at the user level are -

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager	
		Resource Manager Release 1 Specification	
<b>EUROTHERM</b> © Copyright 1991 Eurotherm Limited	<b>EI</b>	DOCUMENT NUMBER HP024105	SHT.  20

```
help_command ::= 'help' | 'help' command_name
```

help on its own prints out brief help on all available commands. help with a command\_name prints out more detailed information on that command.

```
connect_command ::= 'connect' ''' [ resource_name ] ':' task_name ''' |  
                    'connect' ''' [ resource_name ] ''' task_number
```

which connects the debugger to the specified Resource and task. If the Resource name is not specified the local Resource is used.

The debugger may be connected to many tasks. The last task connected is the one that receives any commands typed in, but debug output may be received from any previously connected task.

It is possible to connect the debugger to itself to debug the local Resource.

```
disconnect_command ::= 'disconnect' ''' [ resource_name ] ':' task_name ''' |  
                       'disconnect' ''' [ resource_name ] ''' task_number
```

This disconnects the debugger from the specified task.

```
ping_command ::= 'ping' ''' [ resource_name ] ':' task_name ''' |  
                 'ping' ''' [ resource_name ] ''' task_number
```

can be used to find whether the specified Resource and task exists on the network. A ping message is sent and if the destination is found a ping acknowledge message is returned.

```
whatis_command ::= 'Whatis' hierarchic_name | 'whatis' hierarchic_name  
                  | 'Whatis' ''' | 'whatis' ''''
```

The whatis command prints out information on the specified hierarchic\_name which is any valid ST name. It prints out the type of the name and array dimensions if found, or an error message if the name is not found by the task the debugger sent the message to (i.e in the Resource the debugger is connected to). Note the name must be the name of a variable or block instanced in the Resource, not the name of a type of block. The capitalised form prints out the names, types, modes and dimensions of all the children if the object referred to is a block, otherwise just the number of children is printed and the block type name. The ''' form is a shorthand for the currently connected Resource name.

```
print_command ::= 'print' hierarchic_name
```

If the referred to object is found and has a single simple value (i.e is not a block or array) the current value is printed. (Array elements may be accessed using the usual [] notation).

```
set_command ::= 'set' hierarchic_name '=' constant_value |  
                'set' ''' hierarchic_name ''' '=' constant_value
```

CONTROLLED DISTRIBUTION COPY  
ONLY IF COLOUR STAMPED ON  
CONTROL SHEET.

**EUROTHERM**

© Copyright 1991 Eurotherm Limited

DOCUMENT  
REVISION

1

**EI**

Resource Manager

Resource Manager Release 1 Specification

DOCUMENT NUMBER

HP024105

SHT.

21

attempts to set the specified name to the specified constant value. If the name is an object of simple type (i.e not an array or block) and the type matches that of the value, that value is written, and the previous value returned, otherwise an error message is returned. `constant_value` is any valid literal ST value. (Array elements may be accessed using the usual `[]` notation).

```
break_command ::= 'break' [ 'entry' | 'exit' | 'either' ] '
                  [ 'task' | hierarchic_name ]
```

sets a break point. Execution of the connected task is suspended when the break point is reached. The break point is set at entry, exit or either of execution of the specified function block instance or of the whole task if 'task' is used. The function block must have been compiled with the debug option on the ST compiler.

One of 'entry', 'exit', or 'either', must be present, One of 'task' or hierarchic\_name must be present.

→ break at 'ST line' supported, & stop command

```
continue_command ::= 'continue'
```

continues execution from a break point.

```
delete_command ::= 'delete' [ 'entry' | 'exit' | 'either' ] '
                    [ 'task' | hierarchic_name ]
                    | delete 'all'
```

deletes the specified break point or all break points.

```
ref_command ::= 'ref' ref_name '=' reference_string
```

sets up a reference tagged by name `ref_name` to a set of objects. The `reference_string` is an expandable string of the same form as the `ref` property of `vrefs`, except that no local names can be specified (because there is no local block to match to!) and that no Resource specification may be supplied because the debugger sends all messages to the connected Resource and task.

The debugger sends a read template message to the connected task, and if the objects all exist then the set of specified objects can then be read and written as a group. There is no requirement that the objects belong to any particular task — the task that the debugger is connected to will do reads or writes regardless of which task owns the objects.

```
read_command ::= 'read' ref_name
```

reads the objects in a previously set up reference `ref_name`. A read message is sent to the connected task. The list command can be used to see the last values read (see below).

```
store_command ::= 'store' ref_name '.' element_no '=' constant_value
                 | 'store' ref_name '.' element_no '[' index ']' '=' constant_value
```

stores a value in the element number `element_no` of the local image of the reference `ref_name`. The `index` is used for references that are arrays (multi-dimensional arrays are flattened into single dimensional arrays as specified in section 4.1.1).

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager  Resource Manager Release 1 Specification	
EUROTHERM  © Copyright 1991 Eurotherm Limited	EI	DOCUMENT NUMBER HP024105	SHT.  22

`write_command ::= 'write' ref_name`

writes the values stored in the local image of ref `ref_name` to the actual objects referenced. Note no checks are made on whether outputs of blocks are being written.

`unref_command ::= 'unref' ref_name`

deletes a previously defined reference with name `ref_name`.

`exercise_command ::= 'exercise' times [ 'read' | 'write' ] 'ref_name'`

will read or write the specified reference times times, and report on the number of seconds taken.

`define_command ::= 'define' macro '=' '' substitute ''`

defines the string macro as a shorthand for substitute, which can be any string at all. (Typically this is used to abbreviate commands or long ST names).

`undefine_command ::= 'undefine' '' macro ''`

undefines a previously defined macro.

`list_command ::= 'list'`  
`[ 'breaks' | 'refs' | 'ref' ref_name | 'macros' ]`

lists current break points, or defined references, or the values and types of reference `ref_name`, or currently defined macros.

`redirection_command ::= 'transcribe' name | 'transcribe' '' name''`

copies all commands types to the named file.

`output_command ::= '>' name | '>'`

writes all debug output to the file name; if name is absent debug output is reset to the terminal.

`input_command ::= '<' name | '<'`

means all debug commands are read from the file name; if name is absent debug input is reset to the terminal.

The redirection commands may be useful for loading in pre-defined macros save in a file, and also for setting up regression tests. Two other commands are also useful for regression testing —

`sleep_command ::= 'wait' time | 'pause' time`

which cause the debugger to pause for the specific number of seconds. `wait` causes a sleep once only, whereas `pause` makes the debugger sleep between every command. `pause 0` resets the debugger not to sleep between commands.

Finally the command quit exits the debugger.

CONTROLLED DISTRIBUTION COPY  
ONLY IF COLOUR STAMPED ON  
CONTROL SHEET.

DOCUMENT  
REVISION

1

Resource Manager

Resource Manager Release 1 Specification

**EUROTHERM**

© Copyright 1991 Eurotherm Limited

**EI**

DOCUMENT NUMBER

HP024105

SHT.

23

## 8.2 Error monitoring blocks

There will be a function block provided which will output diagnostic statistics about all the Var References in a task. Its outputs are counters of errors that have occurred in vrefs.

The function block will consist of counts of events ( possibly fleeting ) that are visible from ST and others which are not.

The function block ( Figure 8.1 ) will be of the form :

Inputs :

**OutputMode** Change the mode of the outputs. Valid modes are :

- 0 Display running total
- 1 Display running total since last ZeroRelative
- 2 Display total in last completed CountPeriod

**ZeroAbsolute** Set all values to zero. This has a global effect on all instances of this block.

**ZeroRelative** For mode 1, outputs are now differences from now.

**CountPeriod** Period for mode 2.

Outputs :

**Requests** Number of requests issued

**ReqMatch** Number of read template requests issued

**ReqRead** Number of read requests issued

**ReqWrite** Number of write requests issued

**Responses** Number of successful responses received

**ResMatch** Number of read template responses received

**ResRead** Number of read responses received

**ResWrite** Number of write responses received

**State errors** Number of times a VarRef falls into an errored state.

**MatchError** Number of times a vref falls into a error state after a read template.

**ReadError** Number of times vref falls into error state after a read.

**WriteError** Number of times vref falls into an error state after a write.

**Timeouts** Request timeouts

**MatchTimeout** Number of times a read template has not received a response in the timeout period.

**ReadTimeout** Number of times a read has not received a response in the timeout period.

**WriteTimeout** Number of times a write has not received a response in the timeout period.

**Operation errors** Var Ref operation errors. These are errors that prevent the request actually being sent.

**OpInProgress** Current operation still in progress on a vref when another operation was requested

**BadState** Vref was in the wrong state to perform the requested operation.

CONTROLLED DISTRIBUTION COPY  
ONLY IF COLOUR STAMPED ON  
CONTROL SHEET.

DOCUMENT  
REVISION

1

Resource Manager

Resource Manager Release 1 Specification

**EUROTHERM**

© Copyright 1991 Eurotherm Limited

**EI**

DOCUMENT NUMBER  
HP024105

SHT.

24



**OOTFull** The Outstanding Operation Table (OOT) was full, and therefore the request was rejected.  
Each request requires a free entry in an internal table, the OOT, until the response is received or the request terminates in an error or timeout.

**NoBuffers** No CMS buffers were available for the request.

**ParseFail** Syntax error in ref string. ( The number of occurrences of the status ParseFail ).

**ResolveFail** Error in contents of ref string. ( The number of occurrences of the status ResolveFail ).

#### Status errors ( Section 4.1.5 )

**Mismatch** Template mismatch. ( The number of occurrences of the status TemplateMismatch ).

**Unreachable** Router is unreachable. ( The number of occurrences of the status Unreachable ).

**BadStatus** Status not OK in a received message. ( The number of occurrences of the status BadStatus ).

**ManyOwners** A vref resolves to items held owned by different tasks. ( The number of occurrences of the status NonUniqueOwner ).

#### System errors ( Should not happen ).

**SystemError** ( The number of occurrences of the status SystemError ).

Some status's are not included because they may be implied from others. These are :

**Ok** This is 1 + ResMatch + ResRead + ResWrite

**InProgress** This is a fleeting condition that occurs once a request is issued. This is ReqMatch + ReqRead + ReqWrite

**NoResources** This is NoBuffers + OOTFail.

For every request that is issued one of the following will result :

- A valid response.
- An operation error. The request has not been sent.
- A timeout. No response was received to a request.
- A state error + status error. A response was received but the response actions were not completed due to an error.
- A state error + system error

Therefore :

Completed Requests = Requests - Uncompleted Requests

Completed Requests = Responses + Operation Errors + State Errors + Timeouts

State Errors = Status Errors + System Errors

Uncompleted requests consist of :

- A single outstanding request per VarRef
- Requests aborted by changing the VarRef Ref string.

If any of the request counters exceeds , then the counter will be globally reset, as if the input ZeroAbsolute has been triggered.

CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION  1	Resource Manager	
		Resource Manager Release 1 Specification	
<b>EUROTHERM</b> © Copyright 1991 Eurotherm Limited	<b>EI</b>	DOCUMENT NUMBER	SHT.
		HP024105	25

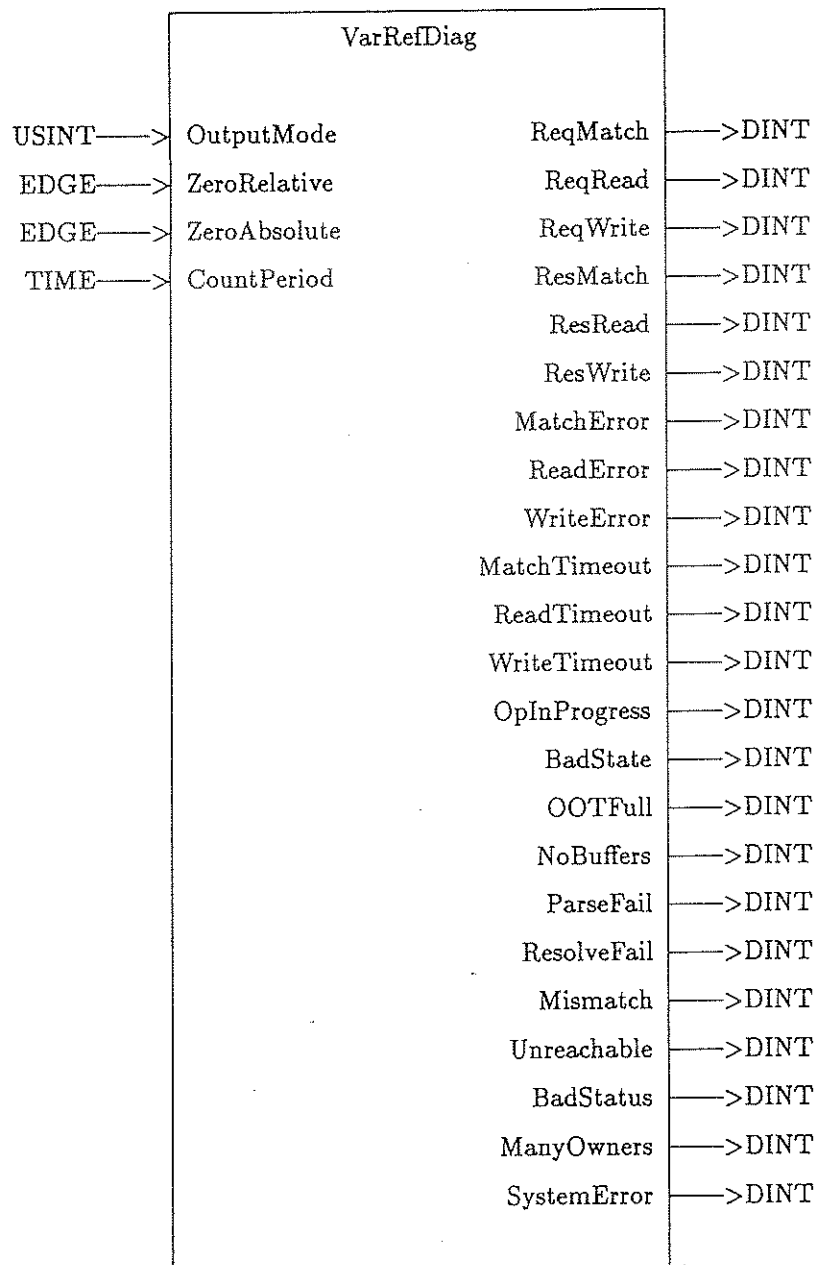


Figure 8.1 Var References Diagnostics

CONTROLLED DISTRIBUTION COPY  
ONLY IF COLOUR STAMPED ON  
CONTROL SHEET.

DOCUMENT  
REVISION  
1

Resource Manager

Resource Manager Release 1 Specification

**EUROTHERM**

**EI**

DOCUMENT NUMBER  
HP024105

SHT.

26

© Copyright 1991 Eurotherm Limited

## 9 Features for PO 2.5

Production Orchestrator version 2.5 will support the vref construct as specified in section 4.1.

Only one task and one program associated with it will be supported. The user will have a default non-editable Resource specification (section 5) of the form

```
RESOURCE <conf_name> ON cell
```

```
TASK unix;
```

```
PROGRAM <conf_name>_p WITH unix : <conf_name>  
END_RESOURCE
```

where <conf\_name> is the name of the cell configuration. Starting the cell will be equivalent to starting the task unix.

Cell to cell communication is only available using ethernet and udp, (see section 6).

The debugger will be a stand-alone unix program provided with the cell.

Use of the Resource will be enabled by setting the configuration option RESOURCE to YES.

## 10 Features for PO 2.6

Production Orchestrator version 2.6 will support a fully editable Resource specification see section 5.

Only one task can be run on the unix host, and will be denoted by the processor name Unix386. Other tasks may be run on a Computrol card, and will be denoted by the processor name Comp68k, provided the memory limitations of the Computrol card are not exceeded.

The task on the unix host will have no built in task inputs.

The tasks on the computrol card will have the same set of inputs and outputs as specified for the PC3000 tasking system (see [2]), and will run with the same execution semantics.

Both ethernet-udp, and EILIN communication will be available.

It is a requirement that any function blocks developed in the EILIN project (see document [1]) shall run in this environment.



CONTROLLED DISTRIBUTION COPY ONLY IF COLOUR STAMPED ON CONTROL SHEET.	DOCUMENT REVISION	Resource Manager	
	1	Resource Manager Release 1 Specification	
EUROTHERM © Copyright 1991 Eurotherm Limited	EI	DOCUMENT NUMBER	SHT.
		HP024105	27