



**EUROTHERM**

---

# **THE RESOURCE MANAGER**

## **A Guide to Tuning the Resource**

### **User Guide**

---

© COPYRIGHT MCMXCIV EUROTHERM LIMITED

All rights strictly reserved. No part of this document may be stored in a retrieval system, or transmitted, in any form or by any means without prior written permission from Eurotherm Ltd

**HA024105C003 2**

M Fox



## Contents

<b>1</b>	<b>Scope</b>	<b>5</b>
<b>2</b>	<b>Related Documents</b>	<b>5</b>
<b>3</b>	<b>The Resources</b>	<b>5</b>
3.1	Global . . . . .	5
3.1.1	Shared Memory . . . . .	5
3.2	Tasks . . . . .	5
3.2.1	The OOT . . . . .	5
3.2.2	The PST . . . . .	6
3.2.3	CMS . . . . .	6
3.2.4	Message Queue . . . . .	6
<b>4</b>	<b>What Happens</b>	<b>6</b>
<b>5</b>	<b>How to Tune</b>	<b>7</b>
5.1	Global . . . . .	7
5.1.1	Shared Memory . . . . .	7
5.1.2	The OOT . . . . .	7
5.1.3	The PST . . . . .	7
5.1.4	CMS . . . . .	8
5.1.5	MessageQueue . . . . .	8
<b>6</b>	<b>How to Determine What to Tune</b>	<b>8</b>
6.1	The OOT Table . . . . .	8
6.2	CMS Buffers . . . . .	8
6.3	CMS Queue Entries . . . . .	9
<b>7</b>	<b>The Loader</b>	<b>9</b>
<b>8</b>	<b>The TASKs</b>	<b>9</b>

---

<b>9</b>	<b>Tools</b>	<b>9</b>
9.1	reslist . . . . .	10
9.2	resvrf . . . . .	11
9.3	resunld . . . . .	12
9.4	msuspy . . . . .	12
<b>10</b>	<b>An Example Tuning Strategy</b>	<b>14</b>

**VERSION HISTORY**

<b>Version</b>	<b>Date</b>	<b>Changes</b>
1	March 10, 1994	Initial issue
2	December 20, 1994	Update for Version 2.2

## 1 Scope

This document describes how a RESOURCE ( Version 2.2 ) may be tuned, and what criteria should be used to determine what should be tuned.

The document is fundamentally concerned with tuning the Communications Messaging Services ( CMS ) of the RESOURCE, and in particular with how to tune the RESOURCE for the messages to be generated by all the VAR REFERENCES within the network of RESOURCEs.

Tuning is not a sequential, but an iterative process, especially as the tuning of one resource may affect the use of another. It will be necessary to read this document in a similar fashion, as a single pass will not be sufficient to grasp the inter-relation of resources.

Coarse tuning is fairly easy to do and will probably yield results quickly. On the other hand fine tuning is a complex and time consuming process and should only be undertaken if there is seen to be a real need.

## 2 Related Documents

- [1] HA024105C001 A Guide to Var References
- [2] HA024105C002 A Guide to the Resource Debugger
- [3] HA024105C004 A Guide to Tuning CMS
- [4] HA024105C005 A Guide to Setting Up CMS Networks
- [5] HA0214105C007 A Guide to the Resource Loader
- [6] HA024105C008 A Guide to the Resource Task

## 3 The Resources

There are resources requiring tuning at two levels

- The global resources, ie those that affect all TASKs.
- Private TASK resources.

### 3.1 Global

#### 3.1.1 Shared Memory

When the RESOURCE is loaded [5] some shared memory is allocated for the database plus the TASK resources.

### 3.2 Tasks

#### 3.2.1 The OOT

Each TASK has a fixed size Outstanding Operation Table (OOT), the number of entries in the OOT determines the maximum number of outstanding operations ( Read Template, Read/Write or Service ) on VAR REFERENCES ( including dynamic ones ) within the TASK.

For each of these operations there is a timeout. This is the time allowed ( in milliseconds ) between issuing the request ( allocating the OOT entry ) and receiving the response.

### 3.2.2 The PST

Each TASK has a fixed size Pending Service Table (PST), the number of entries in the PST determines the maximum number of pending service requests from VAR REFERENCES ( including dynamic ones ) to this TASK.

### 3.2.3 CMS

The CMS resources consist of private buffers and queue entries. For a fuller description see [3].

### 3.2.4 Message Queue

Each TASK has a message queue into which it buffers all messages received from CMS so that they are processed at the appropriate time during the TASK execution cycle, ie start of TASK, end of TASK and between RLB execution. This queue is of a fixed length and so effectively throttles the number of messages that may be processed at one of these stages in the TASK execution.

## 4 What Happens

This section summarises the processes involved from the time a VAR REFERENCE generates an initial message request to the time the reply is received or the request is aborted.

- The VAR REFERENCE requires an operation to be performed ( Read Template, Read/Write or Service ) and thus requires a message to be sent.
- A buffer from the CMS is requested which is large enough to hold the message, if no buffer is available then the operation has failed.
- An entry in an Outstanding Operation Table (OOT) is requested for this message, if no entry is available then the operation has failed.
- The message is composed and then issued to the CMS to send
- The CMS determines if the message can be delivered to a local TASK, if it cannot then the CMS requests another of its own buffers into which the message is encoded in Universal format and the initial buffer is freed.
- The CMS then attempts to deliver the message to the target CMS ( which will be the local router TASK if the message is in Universal format ).
- If there is a free input queue entry on the destination TASK then this is allocated to the message, and thus the message is delivered. If no queue entry is available then the message is held locally on a waiting queue until an input queue entry becomes available.
- If the message is being delivered by routers then eventually the message is either delivered to the destination TASK input queue by the router local to the destination TASK or the message is lost if it is not possible to deliver it. This last router will deliver the message to the destination process in universal format and the process will then decode it before delivering it.
- The destination TASK then reads its input queue. It then parses the message and gets one of its own CMS buffers large enough to hold the reply. Once it has composed the reply then it sends the response back the way it came via the same mechanisms and frees the initial buffer. If the TASK cannot get a buffer large enough to compile the response, or all its buffers are in use then the request is ignored. The responding TASK does not require an OOT entry.
- The originating TASK then receives ( eventually ) the response and is able to free its OOT entry. If the reply is not received within the timeout period specified by the OOT entry then the response is lost.

## 5 How to Tune

Each resource has a set of default values for all the tunable parameters, which should be sufficient to get most RESOURCES up and running.

This section describes all the tunable resources and how to tune them ( §6 describes how to determine what needs tuning ).

### 5.1 Global

#### 5.1.1 Shared Memory

The default shared memory is only 20000 bytes and is insufficient for many applications. During tuning it is best to allocate as much as is possible, say 500000, and make the shared memory size the last thing tuned, unless it is causing a problem using so much excessive memory.

The **msuspy** tool ( §9.4 ) determines how much memory has been used, and may be used as the argument to the loader [5].

#### 5.1.2 The OOT

By default two OOT entries are allocated for each VAR REFERENCE in each TASK, this will ensure that there will always be enough OOT entries. No OOT is allocated to a TASK without any VAR REFERENCES. If there are large numbers of VAR REFERENCES and it is not envisaged that there will be outstanding operations on all of them at the same time then the number of OOT entries may be reduced, saving on memory.

By using the **-oot** option on the TASK [6] the number of OOT entries for all TASKs may altered from the default.

The default timeouts for VAR REFERENCE operations are :

**Read Template** 1 minute

**Read** 5 seconds

**Write** 5 seconds

These values may be altered by use of the **-tot**, **-tor** and **-tow** [6] option for a TASK ( except the router ).

#### 5.1.3 The PST

By default one PST entry is allocated for every service in a TASK, this will ensure that there are always sufficient PST entries for all services to be pending. No PST is allocated if no services are provided by the TASK.

By using the **-pst** option on the TASK [6] the number of OOT entries for all TASKs may altered from the default.

#### 5.1.4 CMS

By default a buffer distribution of “4:256 4:1024 4:4096” is allocated for a TASK. ( including the Resource debugger [2]) . For a full description see [3].

#### 5.1.5 MessageQueue

By default a message queue of length 32 is allocated.

## 6 How to Determine What to Tune

A RESOURCE and its TASKs may be tuned to according to several criteria :

- To reduce memory usage.
- To reduce delays.

In general a balance will have to be struck.

### 6.1 The OOT Table

Unless there are a large number of VAR REFERENCES or memory is extremely tight, there ought not to be a need to reduce the number of OOT table entries. If the number of entries has been reduced then it is possible to inspect a RESOURCE to determine if requests have not been issued because of a lack of OOT entries ( **resvrf** §9.2 ). If Dynamic VAR REFERENCES are used within a TASK then it is possible that the number of OOT entries may not be sufficient and therefore the number may be increased.

In general it ought not to be necessary to reduce the timeouts unless it is important to determine the non-arrival of a response very quickly. If however the RESOURCE has been tuned to reduce memory usage to the minimum even to the extent of imposing delays on messages then it may be necessary to increase the timeouts to allow for resources to become free to deliver responses. It is possible to determine if any TASK is timing out ( **resvrf** §9.2 ).

### 6.2 CMS Buffers

The distribution of CMS buffers is the most complex area of tuning and several iterations may be required to obtain a very fine tune.

Buffers are required for several reasons :

- Because a TASK has VAR REFERENCES.
- Because a TASK is required to respond to debugger messages during a debug session.
- Because a TASK is required to response to VAR REFERENCE requests from other TASKs



The first reason is easy to identify and to quantify. A tool ( `reslist` §9.1 ) may be used to determine what buffers are required to service all VAR REFERENCES. This however takes no account of where the TASK expects those VAR REFERENCES to be found. If it is to be found inside itself then no buffers are required for Read or Write operations. If only Read operations are required the maximum buffer size for handling that VAR REFERENCE may be less.

The second reason is easy to eliminate if that RESOURCE has no embedded debugging and the ST has not been compiled with any debug option. If however debugging is required then a buffer supplement should be added if the TASK is normally tuned finely. An additional distribution such as "4:256 4:1024" would be good enough for simple message sequences. An alternative approach if memory is limited and in particular if it is intended to debug many TASKs is to add CMS global buffers instead.

The third reason is altogether more difficult and to tune before running would require knowledge of what requests are expected. If a TASK is known to be expected to respond to particular VAR REFERENCES then a buffer distribution could be calculated from the source RESOURCES using the tools ( `reslist` §9.1 ). In general it would be necessary to run the RESOURCE and refine with the aid of the tools the buffer distribution of such TASKs. In the first instance just adding a fixed supplement to all TASKs expected to respond to such requests might be sufficient.

See [3] for a discussion on how to choose buffer distributions.

### 6.3 CMS Queue Entries

See [3] for a discussion of tuning CMS process queue entries.

## 7 The Loader

See [5] for a full discussion of the loader and the options that apply to it which permit tuning.

## 8 The TASKs

See [6] for a full discussion of the TASK and the options that apply to it which permit tuning.

## 9 Tools

The following tools are supplied with the RESOURCE to assist in tuning and debugging. There are a number of tools ( specifically `cmsears`, `cmsspy` and `cmsedit` ) supplied with CMS to assist in the tuning of CMS [3].

Each of the tools also accepts the `-y <Key>` option if more than one RESOURCE is loaded under unix.

```
Buffers for VarReferences ONLY :
Recommend : task task1 -i 2:112
Recommend : task task2 -i 2:112
Recommend : task task3 -i 2:1968
Recommend : task task4 -i 2:1968
Recommend : task task5 -i 2:480
Recommend : task task6 -i 2:480
Recommend : task task7 -i 2:3696
Recommend : task task8 -i 2:3696
```

Figure 1: Example output from reslist of recommended buffers

```
Buffers for VarReferences ONLY :
TASK : task1 ;
  prog1.Remote: ReadTemplate = 397(?)/164, Read = 490/677, Write = 807/361
Recommend : task task1 -i 2:816
```

Figure 2: Example output from reslist for a single task

## 9.1 reslist

**reslist** accepts the following options

<TaskName> Only display this TASK, default is all TASKs.

**-buffer** Displays recommended buffers for all TASKs with VAR REFERENCES. The resulting buffers are sufficient to deal with the worst possible case of outgoing messages.

**-trace** The addition of the **-trace** ( with **-buffer** ) option displays the calculations made for each VAR REFERENCE to allow the recommendation to be tuned according to additional knowledge about the VAR REFERENCES.

**-st** List out the ST in a structured form.

**-gad** List of the ST with its GADs in a flattened form.

Figure 1 shows the output from **reslist -buffer**

Figure 2 shows the output from **reslist task1 -buffer -trace**

This TASK has a single VAR REFERENCE in PROGRAM “prog1” called “Remote”. From the output we conclude that :

- if the VAR REFERENCE is only being read then a buffer of 490 bytes would be sufficient instead of 816.
- **reslist** has deduced that 397 bytes would be required to for a ReadTemplate if the VAR REFERENCE to be read has a reference string of the same size as one required to read the template on “Remote” from another RESOURCE. This may be an under-estimate ( hence (?) ).
- The TASK where the VAR REFERENCE is located requires buffers of 164, 677 and 361 bytes to compose replies.

All the buffer figures assume that the messages are sent in Native format. When a message has to be sent via a router then the message is encoded into Universal format which will be no larger, however a supplement of up 48 bytes may be required.

## 9.2 **resvrf**

This tool displays the diagnostic information for VAR REFERENCES. It may be used on loaded or unloaded TASKs (§9.3).

It may be invoked by one of two methods.

**resvrf resource** To display all statistics for all TASKs

**resvrf task** <TaskName> to display statistics for an individual TASK.

Four sets of figures are obtainable from within **resvrf** with the following options :

**-diag** diagnostics for all VAR REFERENCES ( Figure 3 )

**-oot** size and usage of the OOT ( Figure 4 )

**-list** a list of all VAR REFERENCES and the current values of all properties. ( Figure 5 )

**-pst** size and usage of the PST ( Figure 9.2 )

**-brief** Produce a short form of the above ( **-list** and **-diag** only ).

**-full** Produce a fuller for of the above ( **-list** only ).

The most useful of these for tuning is the **-diag** option ( the default ). If a statistic is not displayed then its value is zero. The statistics useful for tuning are :

**MatchTimeout** The number of times a Read Template timed out.

**ReadTimeout** The number of times a Read timed out.

**WriteTimeout** The number of times a Write timed out.

**ServiceTimeout** The number of times a Service timed out.

**OOTFull** The number of times no OOT entry was available. This will only occur when the maximum number of OOT entries has been limited.

**NoBuffers** The number of times an operation failed because the CMS could not provide a buffer large enough.

Figure 3 shows a sample output from **resvrf task task1 -diag**. It could be deduced from the output that there have been 1586 successful writes, 1 write which timed out, 1 read template which completed, 1 which timed out, and 1 in progress. The implication is that there is a single VAR REFERENCE on this TASK to a TASK that has become unreachable.

The line giving the Requests summary is a sum of the form

**Req = Res + OpErr + StatErr + Timeout + Cancel + Unsuc = Complete**

where

**Req** The total number of requests made.

**Res** The total number of responses received.

**OpErr** The total number of operation errors. ( InProgress + BadState + OOTFull + NoBuffers + ParseFail + ResolveFail ).

---

```

      VAR REFERENCE Diagnostics for "task1"
ReqMatch      =      3
ReqRead       =      1
ReqWrite      =    1587
ResMatch      =      1
ResRead       =      1
ResWrite      =    1586
MatchTimeout  =      1
WriteTimeout  =      1
      Summary
Incomplete requests =      1
Requests           =    1590 = 1587 + 0 + 0 + 2 + 0 + 0 = 1589

```

Figure 3: Example output from `resvrf`

```

      OOT Diagnostics for "task1"
Size      =    10
Free entries =    10
Out entries =     0
      Timeouts
Template = T#1m
Read     = T#5s
Write    = T#5s

```

Figure 4: Example output from `resvrf`

**StatErr** The total number of status errors. ( `MatchError` + `WriteError` + `readError` + `ServiceError` ).

**Timeout** The total number of timeouts.

**Cancel** The total number of cancelled operations.

**Unsuc** The total number of thruing operations that were unsuccessful.

**Complete** The total number of operations that have run to completion.

### 9.3 `resunld`

`resunld` can be used to unload a whole `RESOURCE` or individual `TASKs`. It operates in one of two modes.

`resunld resource` Unload all `TASKs` in the `RESOURCE` and the `RESOURCE` database.

`resunld task <Name>` Unload `TASK <Name>` only.

`resunld` also accepts the following options

`-quiet` Suppress informational messages.

### 9.4 `msuspy`

The `msuspy` tool can be invoked to determine how much shared memory has been used. This may then be used as the `-size` parameter to the loader.

Figure 7 shows sample output from `msuspy -g resource` showing a sized `RESOURCE`.

---

```
VAR REFERENCE Listing for "task1"

[3.0.36]
^ref           = 'prog1.v'
^resolution    = 1 (Ultimate)
^scan          = T#0s
^timeStamp     = DT#1993-03-03-14:39:06
^qtimeStamp    = QT#0ms
^status        = 0 (Ok)
^readStatus    = 0 (Ok)
^writeStatus   = 3 (Undefined)
^servStatus    = 3 (Undefined)
^dontWrite     = 0
^propertyProtect = 1
^propertyProtect = 1
^newRead       = 1
^relax         = 0
^coherence     = 1
^coherent      = 1
^detectDataChange= 0
^dataChanged   = 0
```

Figure 5: Example output from resvrf

```
PST Diagnostics for "task1"
Size           =    7
Free entries   =    7
Out entries    =    0
```

Figure 6: Example output from resvrf

```
The Shared Memory Spy Tool, Version 1.2
(c) Copyright 1992, 1993, 1994 Eurotherm Controls Limited
Resource shared memory
  26928 bytes starts at 0x12000000
    0 bytes free from 0x12006930
  26928 bytes used
```

Figure 7: Example output from msuspy

## 10 An Example Tuning Strategy

The example tuning strategy is a coarse tune but should yield good results provided the runs are representative. If at the end of this the memory usage is too high then a balance between speed and size will have to be struck or a fine tune made.

- Load the RESOURCE.
- Use `reslist -buffer` to determine the initial set of buffers for all TASKs.
- Run all RESOURCEs for a representative period.
  - Use `cmsspy` [3] to inspect all TASKs and eliminate any unused buffers.
  - Inspect the VAR REFERENCE statistics `resvrf resource`. For all TASKs showing a NoBuffers statistic use `cmsspy` [3] to inspect the buffers. Add some buffers of maximum size to all these TASKs.
- Run all RESOURCEs for a representative period then inspect each RESOURCE using the `cmsears` and `cmsspy` tools [3]. For each TASK holding up delivery of buffers add queue entries with `cmsedit` [3].
- With all TASKs ( suitably modified ) loaded size the memory with `msuspy`.

