



EUROTHERM

THE RESOURCE MANAGER

A Guide to the Resource Loader

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

HA024105C007 1 M Fox



Contents

1	Scope	4
2	Related Documents	4
3	Introduction	4
4	Specifying the RESOURCE	4
4.1	Output Specification	5
4.2	RESOURCE Name	5
4.3	VAR_ACCESS	5
4.4	TASKs	6
5	Shared Memory	6
6	Optimizations	7
7	Database Hiding	7
8	Trace	7
9	Post Loading	7
9.1	Resource Level Wiring	8
9.2	Resource Initialisation	8
9.3	VAR_ACCESS	8
A	Summary of Loader Options	9

VERSION HISTORY

Version	Date	Changes
1	December 20, 1994	Initial Issue

1 Scope

This document describes the Resource loader (Version 2.2).

2 Related Documents

- [1] HA024105C001 A Guide to Var References
- [2] HA024105C003 A Guide to Tuning the Resource
- [3] HA024105C005 A Guide to Setting Up CMS Networks
- [4] HA024105C007 A Guide to the Resource Task

3 Introduction

The resource loader is the resource manager facility that creates a **RESOURCE** database. The loader creates the **RESOURCE** database from one of the following sources :-

- A linked ST compiler output.
- A **RESOURCE.ST** source file specifying the resource level objects (**RLOs**) together with a number of libraries in the form of compiled **RESOURCES**.

The resource loader runs on 3 distinct platforms :-

XEM - The Eurotherm Motorola 68XXX family XEM (for embedded systems)

unix - Interactive Unix

Windows - Microsoft Windows

There are differences between these platforms which are *highlighted* in this document.

It is required that a CMS is already loaded prior to loading the **RESOURCE**.

4 Specifying the RESOURCE

The specification of the **RESOURCE** may be one of the following :-

- The default compiled **RESOURCE**. This is the default unless other options are given to the loader.
- An alternative compiled **RESOURCE**. This is given as a single argument to the loader. *For Windows this is the name of a DLL. The .DLL may be dropped from the DLL name. For XEM this is the entry point of a compiled RESOURCE. It is not normally specified by hand.*
- An interpreted **RESOURCE** specification in a file (see Figure 1) (in IEC1131-3 **RESOURCE** declaration format) given by the **-input** option, together with a number of libraries of **RLOs** given by the **-library** option (the default one being the default **RESOURCE**). The libraries are searched in the order in which they are specified on the command line with the default library being searched last. Some of the specification in an input file may not be applied by the loader and needs to be extracted using the **-output** option and then applied later (S 9).

This **RESOURCE** specification may then be extended in some of the following ways :-

- Additional **TASKs** (without associated **ST**) may be added using the **-extra** option.

```
RESOURCE regress18 ON Unix
TASK task1 ;
TASK task2 ;
FUNCTION_BLOCK block WITH task1 : fbreg18 ;
FUNCTION_BLOCK REFERENCE vblock WITH task2 : fbreg18 ;
FUNCTION_BLOCK wblock WITH task1 : fbreg18 ;
END_RESOURCE
```

Figure 1: Example RESOURCE Text Specification

```
The Resource Loader, Version 2.2 Development
(c) Copyright 1992, 1993, 1994 Eurotherm Controls Limited
RESOURCE "regress2" loaded
Memory: 15772 bytes of 20000 used
      2320 bytes of instance data
0 Var References
```

Figure 2: Example loader output

4.1 Output Specification

If the `-input` option is used to load a **RESOURCE** from a text file then part of that specification is applied after loading (§9). If the `-output` option is used with `-input` then these specifications are output in separate files, all having the same base name but with different extensions. These file extensions and their contents are listed below :-

- .RLW** Resource level wiring. This may be used as input to `resinst` §9.1.
- .RIF** Initialisation of RLO **INPUTs**, setting of **GLOBAL** data and initialisation of **VAR REFERENCE** properties. This may be used as input to `resinst` §9.2.
- .VAC** Application of **VAR_ACCESS** specifications. This may be used as input to `resvac` §9.3.

4.2 RESOURCE Name

By default the **RESOURCE** is loaded with the name of the specified **RESOURCE**. This name may be changed with the `-name` option. This mechanism allows a **RESOURCE** specification to be instantiated with another name.

4.3 VAR_ACCESS

By default there is provision for 128 **VAR_ACCESS** declarations. This value may be changed at load time with the `-access_paths` option to another value. The number of access nodes given by `-access_nodes` must be greater than or equal to the number of access paths.

4.4 TASKs

The loader may be used to allocate resources used by the TASKs [2] [1] [4] that require shared memory to be allocated. These are :-

- The outstanding operation table (OOT).
- The pending service table (PST).
- The resource messaging queue.
- The resource proxy TASK.

Whenever the `-automatic` option is used all the above are allocated with the default values.

It is possible to allocate each of the above on a per TASK (or per RESOURCE) basis using the `-alloc` option. Each option is given as `-alloc option[:<TaskName>][=<Arg>]`. If `<TaskName>` is not given then the allocation applies to all TASKs in the RESOURCE. The table below summarises the options :-

Description	Option	Default Arg
OOT	oot	Default for the number of VAR REFERENCES
PST	pst	One per SERVICE
Resource Messaging Queue	rmq	32
Proxy TASK	Complexity	200
All of the above	all	No argument.

5 Shared Memory

The RESOURCE database is loaded into an area of shared memory. The loader allocates this area of shared memory. The size of this area of shared memory is determined in one of the following ways :-

- The default value of 20000 bytes is used. This is not usually an appropriate choice.
- A specific value (in numbers of bytes) may be specified using the `-size` option.
- The `-automatic` option may be used to allocate a large area of memory (*1M on unix, 256k on Windows, the maximum available on XEM*) and the unused portion of memory is freed (*not on unix*) on completion of loading. When this option is used all the shared memory structures that may be allocated by the TASKs must be allocated by the loader (§4.4).

6 Optimizations

The resource loader has the facility to apply (or not apply) a number of optimizations with the `-optimize` (or `-nooptimize`) options.

Description	Option	Supported By		
		XEM	Unix	Win
Optimize template strings by referring back to the compiler output.	c	Yes	No	Yes
Generate only one copy of identical GAD tables. ie there is more than one instance of the same RLO.	g	Yes	Yes	Yes
Generate only one copy of the same template string	s	Yes	Yes	Yes
Destroy compiler output by writing the resource database structures on top of the compiler output.	t	No	No	No

The option `*` is equivalent to all the above options.

The default is that all optimizations applicable to the target are made.

7 Database Hiding

The loader provides the facility to hide selective parts of the database on loading. This has the effect of reducing the size of the database and thus making those items that have been hidden unaddressable. The following categories of data may be hidden with the `-hide` option :-

Description	Option
Internals variables	i
SFC STEP bodies	s

The option `*` is equivalent to all options.

8 Trace

The loader has the facility to trace the loading process. This is primarily for debugging the loader itself but may be used to determine if any errors are present in the compiler output. The trace options is specified with the `-t` option together with either or both of the `c` and `m` qualifiers. These specify tracing of calls to the loader methods and addresses at which objects are loaded into shared memory respectively.

9 Post Loading

This section describes the procedures and methods required to complete the loading process that are not actually performed by the loader itself nor are applied by the **TASKs** themselves.

In particular this section describes how the loader output files (from the `-output` option may be applied.

```
wblock.In := block.Out
vblock.In := block.Out
```

Figure 3: Example .RLW contents

```
task1.INTERVAL := T#100ms ;
wblock.In := 7 ;
vblock^scan := T#10s ;
vblock^ref := 'block' ;
```

Figure 4: Example .RIF contents

9.1 Resource Level Wiring

In order to apply resource level wiring the resinst tool is used. It is assumed that the wiring statement are contained in a file (.RLW). These may be applied using the command :-

```
resinst -input <File>
```

The current status of the resource level wiring may be extracted using the command :-

```
resinst -input <File> -writeprotect -assign
```

9.2 Resource Initialisation

In order to apply the instance based initialisation values the resinst tool is used. It is assumed that the wiring statement are contained in a file (.RLW). These may be applied using the command :-

```
resinst -input <File>
```

The current status of the initialisation may be extracted using the command :-

```
resinst -input <File> -writeprotect -assign
```

9.3 VAR_ACCESS

In order to allocate VAR_ACCESS paths the resvac tool is used. It is assumed that the VAR_ACCESS paths are contained in a file (.VAC). These may be applied using the command :-

```
resvac -create -input <File>
```

The VAR_ACCESS paths may be extracted using the command :-

```
resvac -list
```



```
tag_in : wblock.In : DINT READ_WRITE ;
tag_out : wblock.Out : DINT READ_ONLY ;
```

Figure 5: Example .VAC contents

A Summary of Loader Options

The loader is invoked as either `loader` or `loadtask loader`. The applicable options are :-

- `<Resource>` - Specifies the resource to be loaded. (§4)
- `-access_paths <Num>` - Specifies the number of `VAR_ACCESS` paths. (§4.3)
- `-access_nodes <Num>` - Specifies the number of access nodes (one used by each `VAR_ACCESS`). (§4.3)
- `-automatic` - Automatic loading and allocation of `TASK` resources and subsequent freeing of unused of shared memory. (§5)
- `-extra <TaskName>[:<Processor>` - Add additional `TASKs` to the `RESOURCE` without any associated `ST`. (§4)
- `-t[c [m]]` - Trace the loading process. (§8)
- `-size <Size>` - Specifies the size of the shared memory to be allocated for the `RESOURCE` database. (§5)
- `-name <Name>` - Specifies the `RESOURCE` name. (§4.2)
- `-optimize [c|g|s|t|*]` - Specifies optimizations to be applied. (§6)
- `-nooptimize [c|g|s|t|*]` - Specifies optimizations not to be applied. (§6)
- `-hide [i|s|*]` - Selectively hide part of the `RESOURCE` database. (§7)
- `-input <FileName>` - Specify the input file for a `RESOURCE` specification. (§4)
- `-output <BaseFileName>` - The base name for output files from a `RESOURCE` load. (§4.1)
- `-library <Name>` - Specifies a library of `RLOs`. (§4)
- `-alloc [all|oot|pst|rmq|proxy [:<TaskName>][=<Arg>]` - Specifies `TASK` shared memory allocation options. (§4.4)
- `-quiet` - Do not output loader copyright message and load statistics.
- `-y <Resource Key>` - The Resource IPC key.
- `-yCMS <CMS Key>` - The CMS IPC key.

