RTA-TRACE

Report Mark-up Language Reference Guide

Contact Details

ETAS Group

www.etasgroup.com

Germany

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

Tel.:+49 (711) 8 96 61-102
Fax:+49 (711) 8 96 61-106

www.etas.de

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

Tel.: +1 (888) ETAS INC
Fax: +1 (734) 997-94 49

www.etasinc.com

Japan

ETAS K.K.
Queen's Tower C-17F,
2-3-5, Minatomirai, Nishi-ku,
Yokohama, Kanagawa
220-6217 Japan

Tel.: +81 (45) 222-0900
Fax: +81 (45) 222-0956

www.etas.co.jp

France

ETAS S.A.S.
1, place des États-Unis
SILIC 307
94588 Rungis Cedex

Tel.: +33 (1) 56 70 00 50
Fax: +33 (1) 56 70 00 51

www.etas.fr

Korea

ETAS Korea Co. Ltd.
3F, Samseung Bldg. 61-1
Yangjae-dong, Seocho-gu
Seoul

Tel.: +82 (2) 57 47-016
Fax: +82 (2) 57 47-120

www.etas.co.kr

Great Britain

ETAS UK Ltd.
Studio 3, Waterside Court
Third Avenue, Centrum 100
Burton-upon-Trent
Staffordshire DE14 2WQ

Tel.: +44 (0) 1283 - 54 65 12
Fax: +44 (0) 1283 - 54 87 67

www.etas-uk.net

Copyright

Document TD000010-002

Contents

1 About this Manual

RTA-TRACE is a software logic analyzer for embedded systems. It provides the embedded application developer with a unique set of services to assist in debugging and testing a system. Foremost amongst these is the ability to see exactly what is happening in a system at runtime with a production build of the application software.

Using the Report Plug-in for the RTA-TRACE client, custom reports can be generated from the trace data captured from the target application. This manual describes the RTA-TRACE Report Mark-up Language that allows these reports to be customized.

1.1 Who Should Read this Manual?

It is assumed that you want to produce custom reports from the data captured by RTA-TRACE. It is also assumed that you have set up and run RTA-TRACE according to the *Getting Started Guide* and/or the *User Reference Manual*.

1.2 Document Conventions

Important: Notes that appear like this contain important information that you need to be aware of. Make sure that you read them carefully and that you follow any instructions that you are given.

Portability: Notes that appear like this describe things that you will need to know if you want to write code that will work on any target processor.

In this guide, program code, header file names, C type names, C functions and API call names all appear in the `courier` typeface. When the name of an object is made available to the programmer the name also appears in the `courier` typeface, so, for example, a task named Task1 appears as a task handle called `Task1`.

2 Usage

Reports are written to files, and are based on *report templates*. This section describes how the report generation process can be used to generate some example reports. A full description of the available template elements can be found in section 3.

Generation of custom reports requires an additional license feature, available from LiveDevices.

2.1 Generating a customised report

To generate a custom report, it is first necessary to produce a report template. The report template can extract various parameters from system objects, and combine these with fixed text strings as desired.

2.1.1 Creating a report template

A report template is able to programmatically extract information from system objects, generating an output file. Let's start with a simple example:

The following template iterates through every object in the system listing its type, name, and identifier:

```
Objects in this system:
$nl$
$nl$
$repeat:ob=all$
@ob.Type@ "\t" @ob.Name@ "\t" @ob.ID@
$nl$
$~repeat$
```

The output will look something like this (depending upon the system configuration):

```
Objects in this system:

Error          Hook                 65533
Cat2 ISR       SystemISR            16
Task           tskLeader            9
Process        task_tskLeader       10
Task           tskInterfere         7
Std resource   rscContention        2
Std resource   RES_SCHEDULER        1
Counter        SystemTimer          11
Alarm          almLeader            13
…
Tracepoint     interfereValue       32771
Interval       interfereRunTime     40965
Background     (Unallocated)        65531
OS             activity             65530
```

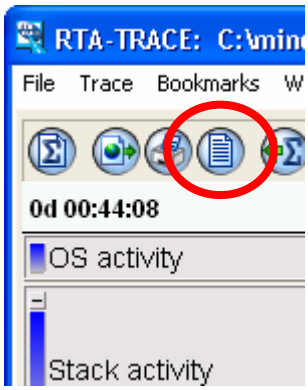
Many report templates can be applied to any system that will be traced (such as the one above). The ability to re-use templates allows engineers to extract common information about each system.

Report templates can be generated in any text editor, as long as the report template is saved as plain ASCII text. The resultant report format is entirely under user control – examples will be given later to produce HTML files (suitable for viewing in a web browser), and CSV (Comma Separated Value) files (suitable for viewing/analysis in a spreadsheet).

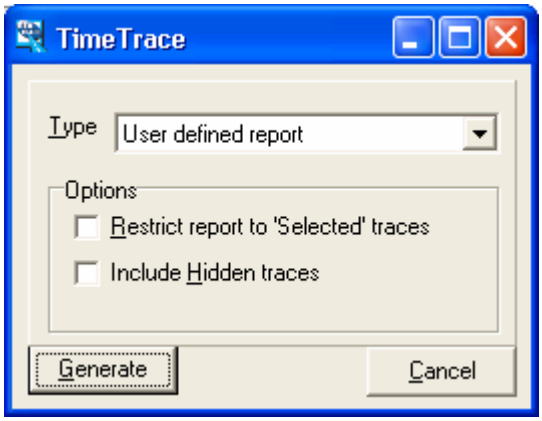
2.1.2 Using a report template

Generation of a report from a template is carried out from the RTA-TRACE TimeTrace Client:

1. With the RTA-TRACE Client running, and displaying trace data (either live or from a file) click the 'Generate Report' button on the toolbar:



2. The following dialog box will then appear, and 'User defined report' should be selected (as shown):



3. A file selector will then appear prompting you for the file containing the report template. Once the template has been found, another file selector will appear prompting you for a filename and location for the generated report.

The generated report can then be viewed in whichever viewer is appropriate for the type of file.

2.1.3 Viewing a report

The correct viewer for a report is necessarily dependent upon the type of report generated. Generally, reports will be viewable in any text editor, but they may make more sense in a different viewer (e.g. CSV files viewed in a spreadsheet, HTML files in a web browser).

2.1.4 Example 1 (HTML)

This example is identical to the plain text example seen in section 2.1.1, except for the addition of HTML tags:

```
<HTML>
<HEADER>
<TITLE>Objects in this system</TITLE>
</HEADER>
<BODY>
<H1>Objects in this system</H1>
$repeat:ob=all$
<P> @ob.Type@ "\t" @ob.Name@ "\t" @ob.ID@ </P>
$nl$
$~repeat$
</BODY>
</HTML>
```

The HTML headers are simply inserted as they are required to appear in the output.

Of course with a little HTML knowledge, it is possible to put this information into a table, apply text formatting etc.

2.1.5 Example 2 (HTML)

This example shows how a conditional might be used to affect the generated report. Again, each object in the system is shown, but this time those objects with IDs greater than 32767 have their names highlighted in bold text.

The generated output is displayed in a tabular format.

```
<HTML><HEADER>
<TITLE>Objects in this system</TITLE>
</HEADER>
<BODY>
<H1>Objects in the system @System@</H1>
<TABLE>
$nl$
$repeat:ob=all$
<TR><TD>
$if:@ob.ID@,gt,32767$<B>$~if$  // start BOLD
@ob.Type@
$if:@ob.ID@,gt,32767$</B>$~if$ // end BOLD
</TD>
<TD> @ob.Name@ </TD>
<TD> @ob.ID@   </TD>
</TR>
$nl$
$~repeat$
$nl$
</TABLE></BODY></HTML>
```

2.1.6 Example 3 (CSV)

The trace data captured by RTA-TRACE can also be analyzed using the statistics plug-in, and this statistical data can be captured in a report.

The following example displays a number of key statistics about the tasks in a system. The resultant file can be imported into a spreadsheet and subsequently plotted as a graph (if the spreadsheet program supports such operations).

The following template generates a table of CPU utilization:

```
$fmt:ticks$
$repeat:ob=tasks$
"@ob.Name@ , @ob.Utilization@"
$nl$
$~repeat$
$repeat:ob=isrs$
"@ob.Name@ , @ob.Utilization@"
$nl$
$~repeat$
$nl$
```

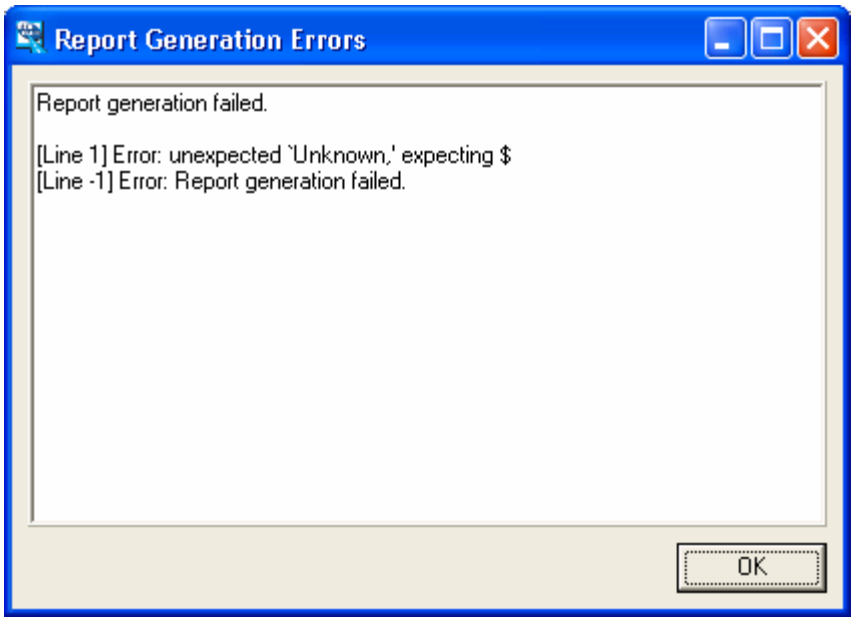
This will generate a CSV file suitable for importing into a spreadsheet. All that is required to do is to generate a figure for the ‘unknown’ time in the

measurement period. This can simply be done by subtracting the sum of utilizations from 100 (For example: In Excel, this may be done by inserting '`=100-SUM(B1:Bn)`' where *n* will be the number of objects in the graph).

As well as using pre-defined statistics such as Utilization, it is possible to define your own statistics within the TimeTrace plug-in, and refer to those in the same way as built-in statistics.

2.2 Errors

In the case of a malformed report template, the Report Generation will display an error dialog something like this:



Error messages indicate the line at which the error occurred – line '-1' above is a non-line-oriented error.

3 Reference

Customised reports consist of the following elements:

- Text
- Comments
- Command tokens
- Variable references
- Object references

The following subsections describe these elements in greater detail.

3.1 Text

The plug-in supports quoted (using " characters) and unquoted text.

The interpretation of command tokens (between $ marks – see below) is suppressed in quoted text.

Note: variable and object references (between @ marks – see below) are interpreted in both quoted and unquoted text.

New-lines, carriage returns and tabs in quoted text are supported and preserved.

Within quoted text, it is possible to suppress the meaning of the '@', '"' and '\' characters by using the '\' character in front of the character. This means that within quoted text, object attribute and statistics references can be prevented and that quote and backslash characters can be included.

The plug-in also supports the insertion of arbitrary 8-bit characters via C-style octal escape sequences within quoted text.

Example 1 Quoted versus unquoted text. The following fragment:

```
This text
" is followed by "
more text
$nl$
"Command tokens such as $nl$ are
ignored in quoted text"
```

Produces:

```
This text is followed by more text.
Command tokens such as $nl$ are
ignored in quoted text
```

Example 2 Escaping of special characters. The following fragment:

```
This text
" \"is followed by\" "
This other text
```

Produces:

```
This text "is followed by" This other
text.
```

Example 3 Object reference. The following fragment:

```
The time is now @now@.
$nl$
"The value of \@now\@ is @now@."
```

Produces:

```
The time is now 12/01/2004 10:57:57.
The value of @now@ is 12/01/2004
10:57:57.
```

Example 4 Insertion of arbitrary characters via escape sequence. The fragment:

```
" \101 \102 \103 \n \104 \105 \106 "
```

Produces:

```
 A B C
 D E F
```

3.2 Comments

Comments can be inserted into a report template which will not appear in the generated report. Comments are started by the characters **//** and terminated by a new-line (which forms part of the comment).

Example the following report fragment:

```
This should /app//remove this
ear/.
```

Generates the following output:

```
This should /appear/.
```

3.3 Command tokens

Command tokens are contained within **$** pairs (e.g. **$nl$**) and give reports the ability to contain programmatic elements. For example, iteration over all tasks in a system (using **$repeat:**...**$**), or some decision based output (using **$if:**...**$**).

3.3.1 font

Note: This token is deprecated, it is preferable in most cases to simply insert text equivalent to the generated content.

Syntax

Started by: **$font:***string***$**

Ended by: **$~font$**

Description

The **$font:**...**$** token inserts the text **<***string***>** into the output stream.

The **$~font$** token inserts the text **</***string***>** into the output stream where the text comes from the matching **$font:**...**$** token.

These command tokens can be used to insert a variety of paired tags as used in HTML (e.g. and , <h1> </h1>, etc.).

The plug-in supports nested **$font:**...**$** command tokens.

Example

The following fragment:

```
$font:p$A paragraph with $font:i$
italic$~font$ text.$~font$
```

produces this HTML output:

```
<p>A paragraph with <i>italic</i>
text.</p>
```

3.3.2 fmt

Syntax **$fmt:***style***$**

Description The **$fmt:...$** token changes the output format for subsequent statistics. The default output format is **str**.

The valid values for *style* are:

- **ms** – output as milliseconds (no units are displayed)
- **ticks** – output as ticks (no units are displayed)
- **str** – output as a string (appropriate units are displayed)

Invalid values of *style* revert to the default.

This token can be repeated as often as is required in a report.

Example The following excerpt generates each of the format types for the user object **A** (see statistic reference section 3.5):

```
"default: time = @A.Time Stat 1@\n"
$fmt:ms$
"ms      : time = @A.Time Stat 1@\n"
$fmt:str$
"str     : time = @A.Time Stat 1@\n"
$fmt:ticks$
"ticks   : time = @A.Time Stat 1@\n"
$fmt:invalid$
"invalid: time = @A.Time Stat 1@\n"
```

Gives the following output:

```
default: time = 2ms 340us 100ns
ms      : time = 2.340100
str     : time = 2ms 340us 100ns
ticks   : time = 23401
invalid: time = 2ms 340us 100ns
```

3.3.3 repeat

Syntax

Started by: **$repeat:***var*=*name*$

Ended by: **$~repeat$**

Description

The **$repeat:**...**$** tag supports the following values for *name*:

- **tasks** and **task** – All tasks (includes OSEK-type tasks and cooperative tasks);
- **isrs** and **isr** – All ISRs (includes Cat0, Cat1 and Cat2);
- **resources** and **resource** – All resources (standard and linked);
- **std_resources** and **std_resource** – Standard resources only;
- **intervals** and **interval** – Intervals;
- **processes** and **process** – Processes;
- **tracepoints** and **tracepoint** – All tracepoints;
- **profiles** and **profile** – All task/ISR profiles;
- **group** and **groups** – All reported groups;
- **all** – All objects.

When processing a **$repeat:**...**$** loop the encapsulated tokens are interpreted once for each item in the object dictionary that matches *name*.

When the repeat loop is interpreted, the variable *var* is set to the object type and name for each matching object. The variable thus created is then available for use in reports via the normal variable-access mechanism.

The report mark-up language supports nested **$repeat:**...**$** loops.

Example

The following fragment produces a list of the types of each object in a system:

```
$repeat:obj=all$
"Object is of type '@obj.Type@'\n"
$~repeat$
```

Producing:

```
…
Object is of type 'Task'
Object is of type 'Task'
Object is of type 'Cat2 ISR'
Object is of type 'Interval'
Object is of type 'Std resource'
Object is of type 'Linked resource'
…
```

3.3.4 for

Syntax

Started by: **$for:***var*=*low*,*high*$**

Ended by: **$~for$**

Description

When processing a **$for:**…**$** loop the plug-in interprets the encapsulated tokens once for each integer in the range [*low*, *high*].

For each interpretation of the encapsulated tokens, the variable *var* is set to the current loop counter value and is available for use in reports via the normal variable-access mechanism.

The plug-in supports nested **$for:**…**$** loops.

Example

This simple fragment:

```
$for:z=1,3$
"Line @z@ "
$for:x=1,@z@$
"*"
$~for$
$nl$
$~for$
```

generates this output:

```
Line 1 *
Line 2 **
Line 3 ***
```

3.3.5 do

Syntax

Started by: **$do:***var*=*objects*$

Ended by: **$~do$**

Description

When processing a **$do:**...**$** loop the plug-in interprets the encapsulated tokens once for each integer in the range [*low*, *high*].

For each interpretation of the encapsulated tokens, the variable *var* is set to the object type and name for each matching object or "unknown object" if the object does not exist.

The plug-in supports nested **$do:**...**$** loops.

Example

The following script iterates all groups defined in the time-trace visualizer and displays their content:

```
Groups:$nl$
$repeat:i=groups$
"  @i@ members = @i.Members@"
$nl$
$do:m=@i.Members@$
"    @m@"
$nl$
$~do$
$nl$
$~repeat$
```

Typical output:

```
Groups:
  Group 1 members = 9,3,7,5,16
   Task tskLeader
   Task tskFollower
   Task tskInterfere
   Task tskIdle
   Cat2 ISR SystemISR
```

3.3.6 if

Syntax

Started by: **$if:<lhs>,<cond>,<rhs>$**

Ended by: **$~if$**

Description

The report mark-up language includes support for conditional text. The tokens encapsulated by the block are only interpreted if the condition triplet evaluates to 'true'.

Conditionals can be nested, both within other conditional blocks as well as within `$repeat$` and `$for$` loops.

Supported values for the *<cond>* field are **eq, ne, gt, lt, le, ge**.

Example

The following fragment will iterate over the objects in a system and display the object type. If the object has an ID of more than 1000, it is displayed in bold (using HTML tags).

```
…
$repeat:ob=all$
$if:@ob.ID@,gt,1000$ //start BOLD
<B>
$~if$
@ob.Type@
$if:@ob.ID@,gt,1000$ // end BOLD
</B>
$~if$
$nl$
$~repeat$
…
```

3.3.7 nl

Syntax

nl

Description

The **nl** token Inserts a Carriage Return + Line Feed (CRLF) pair into the output stream. This differs from simply using `"\n"` .which simply inserts a carriage return into the output stream.

3.4 Variable access

The plug-in supports access to run-time defined variables via the token `@variable@`.

The following variables are defined before token interpretation starts:

- **`delim`** – the delimiter used for CSV reports. This is inherited from the underlying Windows Operating System and is dependent upon the 'locale'. If the locale's decimal separator is a comma, then `delim` is a 'tab' character; otherwise `delim` is a comma.
- **`now`** – the current time formatted according to the current system locale.
- **`start`** – the time of the first record in the report using the "Multiple units" format. If this is also the first record in the buffer then append "(start of trace data)" to the time.
- **`end`** – the time of the final record in the report using the "Multiple units" format. If this is also the last record in the buffer then append "(end of trace data)" to the time.
- **`ns_per_tick`** – the number of nanoseconds represented by one tick.
- **`product`** – the report generator's long name.
- **`version`** – the Window's version number of the Report plug-in DLL.

3.5 Object Attribute and Statistic Reference

Objects are the fundamental components of the system being traced (i.e. tasks, ISRs, resources, etc.). Object attributes are referenced using the ***`@Object.Attribute@`*** mechanism.

Objects can be referenced in the following ways (assuming a task called **`tsk1`**):

- The object's name (e.g. **`tsk1`**);
- The combination of object type and name (e.g. **`Task tsk1`**)
- A variable reference for a repeat or do loop variable[1];
- A synonym (see section 3.6).

For any object ***`Object`*** defined as above, the plug-in supports references to the following attributes:

- **`Name`** – Inserts the Name of ***`Object`*** (e.g. **`@tsk1.Name@`** would produce **`tsk1`** in the output stream);
- **`Type`** – Inserts the Type of ***`Object`*** (e.g. "Task", "Resource", etc.);

[1] Note that this requires a variable reference for a *repeat* loop and not a *for* loop. This is because only *repeat* loops are guaranteed to give an object reference since a *for* loop index is an integer.

- **ID** – Inserts the object identifier of ***Object***.
- **ParentName** – Inserts the name of the parent of ***Object***. If ***Object*** has no parent, **<none>** is inserted.
- **ParentType** – Inserts the type of the parent of ***Object***. If the object has no parent **<none>** is inserted.
- **ParentID** – Inserts the object identifier of the parent of ***Object***. If the object has no parent zero ("**0**") is inserted.
- **Children** – Inserts the number of children objects belonging to ***Object***.
- **Members** – For "groups" only, inserts the member objects of a group as a comma separated list of *object identifiers*.

Example 1

Given a system with a task named tsk1, the following fragment:

```
"Object '@tsk1.Name@' is of type
'@obj.Type@'\n"
```

Produces:

```
Object 'tsk1' is of type 'Task'
```

Example 2

The following fragment:

```
$repeat:obj=all$
"Object '@obj.Name@' is of type
'@obj.Type@'\n"
$~repeat$
```

Produces:

```
…
Object 'tsk1' is of type 'Task'
Object 'tsk2' is of type 'Task'
Object 'isr1' is of type 'Cat2 ISR'
Object 'int1' is of type 'Interval'
Object 'res1' is of type 'Std
resource'
Object 'res2' is of type 'Linked
resource'
…
```

The plug-in also supports references to statistics via the token **@Object.Statistic@**.

The text **Statistic** corresponds to the name that the user has given to a statistic when it is defined.

The text **Statistic** can include object attribute references using '[' and ']' as delimiters. The attribute reference is expanded before the statistic itself is referenced. As an example:

```
$repeat:t=tasks$
  $repeat:r=std_resources$
    "@t.Name@ and resource @r.Name@: "
    $fmt:ms$"@t.Max Net Resource [r.Name]@ ms ("
    $fmt:str$"@t.Max Net Resource [r.Name]@)"
    $nl$
  $~repeat$
$~repeat$
```

Will produce the maximum net resource lock time for each task for each resource in the system – as follows:

```
Task A and resource R1: 0.000400 ms (400ns)
Task B and resource R1: 0.000400 ms (400ns)
ISR Isr1 and resource R1: 0.000400 ms (400ns)
```

Attempting to reference an attribute or statistic that has not already been defined produces an error message in the output file.

3.6 Object Synonyms

Synonyms are provided for objects that only ever occur once in a system but for which the name is not known in advance. The provision of a standard access method for these objects means that they can be more easily referenced in report templates.

The report defines the following synonyms for system objects to be used in object attribute and statistic reference:

- **System** – the OS object.
- **Background** – Unallocated activity (the idle task).
- **Error** – the system error mechanism (hook).

Index

S

Support

For product support, please contact your local ETAS representative.

Office locations and contact details can be found on the ETAS Group website www.etasgroup.com.