Standard Event Message Format (SEMF) Specification

Version 1

Table of Contents

Click on any entry.

Introduction

As the ATE industry matures, many vendors offer networking systems that complement the test systems themselves and help customers get more out of their ATE investment. Many of these networking systems are converging on popular standards such as Ethernet™ to provide a better “fit” with existing systems and increase productivity.

An important shortcoming in this effort has been most testers’ inability to tell the outside world what they are doing so that appropriate action can be taken. To help overcome this problem, Teradyne has developed a simple, flexible and portable mechanism to allow testers to tell other processors on a network what they are doing, and vice-versa. This mechanism is independent of any communication protocol and is described in terms of specific “events” common to a test environment. Called the Standard Event Message Format (SEMF), its specification is contained in this document.

It is our hope that both users and manufacturers of semiconductor ATE will find this standard useful, and will incorporate it into their own operations and products. Teradyne has adopted this standard for all of its UNIX™ operating system based testers. Teradyne derives no direct commercial benefit from developing and propagating this standard, but we hope its usefulness, thoroughness, and full documentation will make Automatic Test Equipment and those who work with it more productive.

Design Objectives

The following objectives provided the basis for this design:

- Define a Standard Event Message Format for passing command, event, and status information between network nodes.
- Allow a test floor monitor to collect and display information on all interesting events on the test floor.
- Allow test systems to request services from other computers.
- Provide a mechanism for a test floor automation system to schedule and initiate functions on the various network nodes.
- Provide a method for the test floor manager to define the action taken when an event message is received.
- Define a simple message format that is easy to implement using a wide variety of communication mechanisms.
- Provide a mechanism for handling user-defined events.
- Make each control and status information message be as complete and as stateless as possible.

In keeping with the philosophy of the open network architecture, the document does not impose any protocol or physical link requirements upon systems which adhere to this specification. The SEMF can be used with any existing communications path that assures correct transmission of data between the two nodes.

Design Overview

The SEMF defines three basic message types required in the ATE networking environment:

- Event messages, allowing test systems to declare significant test floor events and initiate action at a specified node.
- Control messages, allowing factory automation software to schedule and control events on the test floor.
- Status messages, allowing real-time monitoring displays and event logging.

The SEMF defines an event message standard to be used in implementing a network monitoring and control system. Such a system would work as follows:

1. An Event Dispatcher on one node sends an SEMF event message to another node.

2. The Event Dispatcher on the destination node receives the message. (Note that a single Event Dispatcher can both send and receive messages.)

3. The receiving Event Dispatcher examines the message to see if it serves a special function in the monitoring and control system:

   - If it is a **status** message, the Event Dispatcher sends it to a local test floor monitor server. Status messages are discussed below.
   - If it is a **reply** message, the Event Dispatcher sends it to the user process that originally sent the event message requesting a reply.

4. If the message does not serve a special function, its purpose is to trigger an **event handler** program on the local system. The Event Dispatcher initiates the event handler associated with this event. These event handlers are discussed below.

Status Messages

Test floor status tables are maintained by a test floor monitor server, which makes the information available for real-time monitoring displays throughout the network. The tables are maintained for each active network node, and contain information regarding job stations, test heads, and test sites associated with that node. Status information — such as node name, node type, station mode, current part type, and lot, wafer, and part identification codes — will be available for display by the network monitor program.

Event Handlers

An event message may be sent by any network node to any other node, as long as both support an Event Dispatcher. When an Event Dispatcher receives an event message, it initiates the corresponding event handler, which is a program or command sequence that performs some action on behalf of the system that sent the message.

When the Event Dispatcher initiates an event handler, it passes to it any parameters that the event message contained. The handler usually begins by logging the event to an event log file. It can continue by executing one or more commands or programs that perform some function based on the event and the parameters passed to it. The actions of the handler (including the option to perform network event logging) are completely under the control of the FIRMS system manager. If the process that initially sent the event message requested a reply, the event handler constructs and returns a reply message.

Under the VMS operating system, event handlers may be implemented as programs or command files. Under the UNIX operating system, they may be programs or shell scripts. The SEMF specification places no special requirements or restrictions on the kinds of programs that can be used as event handlers. Therefore, there is a wide range of choices for implementations of an Event Dispatcher under other operating systems.

General Message Format

Messages using the Standard Event Message Format are composed entirely of ASCII text. Each message begins with a keyword, the name of the sending node, and a timestamp. Depending on the keyword, some number of additional parameters may follow.

The tokens in the message — the keyword, node name, timestamp, and parameters — are separated by white space, either a space or a tab. Multiple spaces and tabs between parameters are considered valid.

The general format for an SEMF message is:

keyword   source-node   timestamp   [parameters ...]   [RSVP]

The following sections describe the individual tokens.

Keyword

Each event message has as its first token a keyword string that indicates the event type.

If the keyword string begins with a letter, the receiving node attempts to dispatch an event handler with the same name as the keyword. All parameters included in the message are passed to the handler as arguments.

Keywords beginning with a non-alphabetic character are reserved for special functions in the network. Currently, the following special functions are defined:

#	(number sign)	Denotes a test floor status message. The remainder of the keyword specifies the type of status being reported.
!	(exclamation point)	Denotes a reply message to a previously dispatched event message that requested a reply. (See the description of the RSVP parameter.) The remainder of the keyword indicates the type of message being replied to.

All other non-alphabetic characters are reserved for future use.

The Standard Event Message Format permits user-defined keywords for custom network functions. All Teradyne-defined SEMF keywords begin with the characters **t_** in order to distinguish them from any user-defined keywords. Users cannot define keywords beginning with **t_**, **T_**, or a non-alphabetic character. Adherence to this rule will let Teradyne define new SEMF messages without conflicting with user-defined messages already in the field.

Source node

The second token of each SEMF message is the source node, defined as the network node name of the node on which the event occurred (or the node that initiates the message).

Each system in the network has a network node name, assigned by the networking software that connects to the FIRMS system. Multi-CPU systems (such as the A500 and J967 testers) are known by the name of the user computer, as opposed to the name of any CPU that might be used strictly to control test hardware. Testers connected to FIRMS via a Test System Director (TSD) host are identified by a node name composed of the TSD's DECnet node name followed by the TERANET node number as three ASCII digits. This scheme allows FIRMS to support testers connected via multiple TSDs without duplication of node names.

Timestamp

The third token is the timestamp, which is defined as the time when the event took place (as opposed to the time when the message was sent). The time is local to the system on which the event occurred.

The format of the timestamp is:

[d]d-mmm-yy_hh:mm:ss

where:

[d]d	= day of the month (a leading zero can be omitted)
mmm	= first three characters of the month in English
yy	= last two digits of the year
hh	= hour of the day (24 hour clock)
mm	= minute of the hour
ss	= second of the minute

For example, an event occurring at 2:36 PM on September 3, 1987 would have this timestamp:

```
3-SEP-87_14:36:00
```

Parameters

Each message type has as part of its definition the event-specific parameters that follow the three required tokens. SEMF parameters are recognized by position rather than keyword.

Note these points about parameters:

Missing Parameters

If the value for a particular parameter is missing or not available, an underscore (_) is used to indicate the position of the parameter. Trailing underscores may be omitted. For example, in the following message, the underscore indicates that the next-to-last parameter is missing or not available:

```
t_job_load   scafell   9-jun-88_13:56:10   2   dram265k   _   46
```

The following messages are equivalent, because the missing parameters are the final ones, and trailing underscores may be omitted:

```
t_job_load   scafell   9-jun-88_13:56:10   2   dram265k    _    _
t_job_load   scafell   9-jun-88_13:56:10   2   dram265k
```

Embedded Spaces

If a parameter contains spaces, the entire parameter must be delimited by double quotation marks (" ") to indicate that it is a single parameter. For example, in this message, the final parameter is a single text string:

```
t_event_log   blanc   9-jun-88_08:01:23   Mary   "Lot D002534A missing 2 wafers"
```

RSVP Parameter

If the sending process wants a reply to its message, it appends an RSVP parameter to the end of the message. The RSVP parameter has two functions:

- It indicates to the receiving message that a reply is requested.
- It provides the routing information for returning the reply to the initiating process.

The requirements for the RSVP parameter are:

- It must be the final parameter in the message.
- It must begin and end with the at sign (**@**) (The final **@** lets programs quickly test whether a reply is requested, without having to parse the whole message. If the final character of the message string is **@**, then a reply is requested.)
- The internal fields must contain as much routing information as is needed to return a reply to the initiating process.
- Internal fields must be separated by at signs. (Two adjacent **@**s — **@@** — indicate a null field.)

The actual composition of the RSVP parameter — for example, the number of fields and the content of each field — depends on the implementation.

For example, in Teradyne's FILEnet implementation, the initiating process is responsible for delivering the message to the destination node and for retrieving any reply. Routing information is not needed. The RSVP parameter therefore consists only of one field, a message number, which the initiating process uses to match messages with replies. The message number is bracketed by @s, as the RSVP parameter format requires.

Another Teradyne implementation is between two Event Dispatchers, where an initiating process hands an event message to a local Dispatcher, which transmit the message to the appropriate remote Dispatcher. In this case, the RSVP parameter must contain more routing information. As the event message travels — through the user interface, the local Event Dispatcher, the network, and the remote Event Dispatcher — each piece of Event Dispatcher software examines the event message for the presence of an RSVP parameter (that is, a terminating @). If the software finds the parameter, it knows that a reply has been requested, and it appends a further piece of routing information to the parameter. As the reply travels back, each piece of software strips off the last field of the parameter, and use that field to determine in next step in the route back to the initiating process.

General Message Format

In this Event Dispatcher implementation, the RSVP parameter has this format:

@message-number@process-id@connection-id@node-id@network@

where the fields are as follows:

message-number	The message sequence number is an arbitrary number supplied by the initiating process or the user interface. When the reply is returned, this number lets the process match the original message with its reply.
process-id	The Event Dispatcher on the sending node uses this code to identify the initiating process. When it receives the reply, the Dispatcher will deliver it to this process.
connection-id	The Event Dispatcher on the sending node uses this code to identify the interprocess connection on which it received the message. When it receives the reply, it will deliver it on this connection. (Under UNIX, this number might be the file descriptor of a socket connecting the sending process with the Event Dispatcher.)
node-id	The Event Dispatcher on the receiving node uses this code to identify which node requested the reply.
network	The network identifier tells the Event Dispatcher which network to use for sending the reply. Some implementations of the Event Dispatcher may be able to send messages on multiple networks.

System Limitations

While the SEMF specification does not restrict the length of the message or the number of parameters, as a practical matter it is important to take into account any restrictions imposed by the event dispatchers and operating systems involved. At a minimum, event dispatchers must be able to handle messages of at least 200 characters and at least nine tokens (including the keyword).

Event Message Descriptions

Standard Event Message Format messages are designed for general purpose real-time communication between nodes in an ATE network. Messages are identified by keywords. Each Teradyne-supplied event message has a keyword beginning with the string **t_**. Keywords beginning with any other alphabetic character are available for use by customers and applications engineers.

This section contains the detailed message formats for each keyword currently defined by Teradyne in the Standard Event Message Format. Messages for which keywords are reserved are listed below.

Test Floor Event Messages

Node Initialization	t_node_init
Node Termination	t_node_term
Tester Initialization	t_tester_init
Tester Termination	t_tester_term
Operator Login	t_operator_login
Operator Logout	t_operator_logout
Job Load	t_job_load
Job Unload	t_job_unload
Start of Lot	t_start_lot
End of Lot	t_end_lot
Test Data Ready	t_data_ready
Start of Wafer	t_start_wafer
End of Wafer	t_end_wafer
Alarm Message	t_alarm

Equipment Status Messages

Change Mode	t_change_mode
Equipment Failure	t_system_failure
Equipment Restored	t_system_restore

Utilization Messages

CPU Utilization	t_cpu_util
Tester Utilization	t_tester_util
Resource Utilization	t_resource_util

Queries

Are You Alive	t_alive
Ask CPU Utilization	t_ask_cpu_util

Notices

Send Text to Operator	t_opermsg
Send Text to Event Log	t_event_log

Replies

General reply	!t_reply

Alphabetical Listing of Messages

The messages in the section are listed in the order just shown in the previous lists, according to logical groupings. This table lists the messages in alphabetical order, by keyword (all beginning with t_).

Keyword	Message	Page
t_alarm	Alarm Message	24
t_alive	Are You Alive	33
t_ask_cpu_util	Ask CPU Utilization	33
t_change_mode	Change Mode	26
t_cpu_util	CPU Utilization	30
t_data_ready	Test Data Ready	21
t_end_lot	End of Lot	20
t_end_wafer	End of Wafer	23

Event Message Descriptions

Keyword	Message	Page
t_event_log	Send Text to Event Log	34
t_job_load	Job Load	17
t_job_unload	Job Unload	18
t_node_init	Node Initialization	14
t_node_term	Node Termination	14
t_operator_login	Operator Login	16
t_operator_logout	Operator Logout	16
t_opermsg	Send Text to Operator	34
!t_reply	General reply	35
t_resource_util	Resource Utilization	32
t_start_lot	Start of Lot	19
t_start_wafer	Start of Wafer	22
t_system_failure	Equipment Failure	28
t_system_restore	Equipment Restored	29
t_tester_init	Tester Initialization	15
t_tester_term	Tester Termination	15
t_tester_util	Tester Utilization	31

Node Initialization Message (t_node_init)

Function: Used by a network node to declare that it has joined the network.

This message should be sent during the system startup sequence of the tester, trimmer, repair station, or whatever system is sending it. It should not wait for the tester executive software to start up. It should usually be sent as soon as the event dispatcher is running.

Syntax: **t_node_init**

Source Node

Timestamp

Node Type

Operating System Name

Operating System Version Number

Example:

```
t_node_init   katahdin   9-jun-94_02:20:03   SUN_3   SUN_OS   3.2
```

Node Termination Message (t_node_term)

Function: Used by a network node to declare that it is leaving the network, either through an orderly shut down or because its network services are about to be disabled.

This message should be sent during the system shutdown sequence of the tester, trimmer, repair station, or whatever system is sending it. It should be the last message sent before the event dispatcher software shuts down.

Syntax: **t_node_term**

Source Node

Timestamp

Example:

```
t_node_term   katahdin   9-jun-94_14:20:33
```

Tester Initialization Message (t_tester_init)

Function: Used by a tester executive to declare that it is now running.

This message should be sent when the tester executive startup sequence is complete.

Syntax: **t_tester_init**

Source Node

Timestamp

Tester Type

Executive Name

Executive Version Number

Example:

```
t_tester_init   baldy   9-jun-94_02:20:03   a500   image   1.2
```

Tester Termination Message (t_tester_term)

Function: Used by a tester executive to declare that it is shutting down.

This message should be sent at the beginning of the tester executive shutdown procedure.

Syntax: **t_tester_term**

Source Node

Timestamp

Example:

```
t_tester_term   baldy   9-jun-94_14:20:33
```

Operator Login Message (t_operator_login)

Function: The Operator Login message is used by a network node to declare that an operator has just logged in at the tester.

Syntax: **t_operator_login**

Source Node

Timestamp

Job Station Identifier

Operator Name

Notes:

- On some testers, the job number is used for the Job Station Identifier. Some other testers do not have a job number or job station number.

Example:

```
t_operator_login   nevis   9-jun-94_20:43:43   2   karin
```

Operator Logout Message (t_operator_logout)

Function: Used by a network node to declare that an operator has just logged off at the tester.

Syntax: **t_operator_logout**

Source Node

Timestamp

Job Station Identifier

Operator Name

Notes:

- On some testers, the job number is used for the Job Station Identifier. Some other testers do not have a job number or job station number.

Example:

```
t_operator_logout   nevis   9-jun-94_20:47:13   2   karin
```

Job Load Message (t_job_load)

Function: Used by a test system to indicate the name of a new job loaded on that node.

This message should be sent when the job is completely loaded.

Syntax: **t_job_load**

Source Node

Timestamp

Job Station Identifier

Job Name

Test Head(s)

Job Load Time

Notes:

- On some testers, the job number or test head number is used for the Job Station Number. Some other testers do not have the concept of job stations.
- The Job Load Time parameter specifies the number of seconds required to load the job, including vector files and other related data.

Example:

```
t_job_load   scafell   9-jun-94_13:56:10   2   dram265k   1,2   46
```

Job Unload Message (t_job_unload)

Function: Used by a test system to indicate that a previously loaded job has been unloaded on that node.

Syntax: **t_job_unload**

Source Node

Timestamp

Job Station Identifier

Job Name

Test Head(s)

Notes:

- On some testers, the job number or test head number is used for the Job Station Number. Some other testers do not have the concept of job stations.

Example:

```
t_job_unload   scafell   9-jun-94_16:12:09   2   dram265k   1,2
```

Start Of Lot Message (t_start_lot)

Function: Used by a test system to state the lot identification and part type of the lot of parts that is about to begin testing on that node.

This message should be sent when the tester begins testing the first device in the lot.

Syntax: **t_start_lot**

Source Node

Timestamp

Job Station Identifier

Test Head(s)

Job Name

Lot Identification

Sublot Identification

Part Type (or Family ID)

Notes:

- On some testers, the job number or test head number is used for the Job Station Number. Some other testers do not have the concept of job stations.

Example:

```
t_start_lot   vesuvius   9-jun-94_23:59:18   2   1,2   mem1   d002453   3c   m256k001
```

End Of Lot Message (t_end_lot)

Function: Used by a test system to indicate that a lot of parts has completed testing on that node.

This message is sent after the last device in the lot has been tested. This occurs when the operator requests a final summary or gives some other indication that the lot is complete.

Syntax: **t_end_lot**

Source Node

Timestamp

Job Station Identifier

Part Type

Lot Identification

Sublot Identification

Part Count

Yield

Notes:

- On some testers, the job number is used for the Job Station Identifier.
- Part Count is the total number of parts tested in the lot.
- The Yield is defined as the percentage of “good” parts in the lot.

Example:

```
t_end_lot   vesuvius   9-jun-94_03:10:58   2   mem256k   d002453   3c   524   63.2
```

Test Data Ready Message (t_data_ready)

Function: Used to indicate that the test data for a lot of parts is complete and available for inclusion in the database or for use by data analysis software.

This message is sent when the tester executive (or some process under its control) has closed the STDF format test data file and the file is ready for processing.

Syntax: **t_data_ready**

Source Node

Timestamp

Lot Identification

Sublot Identification

Test Data Filename

Notes:

- The Test Data Filename should be a complete path so that the dispatched event handler will be able to locate the data file.

Example:

```
t_data_ready   vesuvius   9-jun-94_03:10:58   d002453   3c   /usr3/stdf/d002453_3c.std
```

Start Of Wafer Message (t_start_wafer)

Function: Used by a tester to indicate the wafer identification of the wafer of devices that is about to begin testing at one test head on that node.

This message should be sent when the tester begins testing the first device on the wafer.

Syntax: **t_start_wafer**

Source Node

Timestamp

Test Head Number

Wafer Identification

Example:

```
t_start_wafer   aconcagua   9-jun-94_11:34:35   1   w1032671
```

End Of Wafer Message (t_end_wafer)

Function: Used by a test system to declare that a wafer of parts has completed testing at one test head on that node.

This message is sent after the last device on the wafer has been tested. This occurs when the operator requests a final summary, the prober signals end of wafer, or some other indication is given that the wafer is complete.

Syntax: **t_end_wafer**

Source Node

Timestamp

Part Type

Wafer Identification

Part Count

Yield

Notes:

- Part Count is the total number of parts tested in the wafer.
- The Yield is defined as the percentage of “good” parts in the wafer.

Example:

```
t_end_wafer   aconcagua   9-jun-94_12:09:07   mem256k   w1032671   124   76.7
```

Alarm Message (t_alarm)

Function: Used by a piece of test floor equipment or software to declare that an alarm condition exists. The message can be used for yield alarms, catastrophic failure alarms, handler jams, calibration failures, utilization alarms, or any number of other conditions in which an operator or test floor supervisor must be notified or which should be logged for future reference or analysis.

Because there are potentially very many types of alarms which can be implemented using this message, there are no restrictions on the values which may be placed in the final four parameters, as long as they conform to the general SEMF specification.

Syntax: **t_alarm**

Source Node

Timestamp

Type of Alarm

Type of Alarming Entity

ID of Alarming Entity

Alarm Level

Examples:

This example shows a yield alarm on a job called “mem256k” on tester “hood,” where the current yield is 52.6%:

```
t_alarm   hood   9-jun-94_10:49:44   yield   job   mem256k   52.6
```

This example indicates a handler jam on a handler called “EG20005” on tester “hood.” The final parameter (Alarm Level) has been omitted:

```
t_alarm   hood   9-jun-94_10:49:44   jam   handler   EG2000_5
```

This example indicates that the percentage of devices in bin 6 has reached 20.3 percent, which is above the threshold level for the specified bin:

```
t_alarm   hood   9-jun-94_10:49:44   percent   bin   6   20.3
```

(continued)

Alarm Message (t_alarm), continued

This example shows a catastrophic failure alarm where 15 consecutive devices have failed for job “mem256k”:

```
t_alarm   hood   9-jun-94_10:49:44   cat   job   mem256k   15
```

This example shows a test failure percentage alarm where 25.1 percent of the devices have failed test number 23:

```
t_alarm   hood   9-jun-94_10:49:44   test_fail_pct   test   23   25.1
```

Change Mode Message (t_change_mode)

Function: Used by a network node to indicate a change of operating mode for some piece of test floor equipment or software. Information collected via this message may be used in generating test floor utilization reports.

Syntax: **t_change_mode**

Source Node

Timestamp

Type

Identifier

Mode

Comment

Notes:

- The Type field indicates what kind of resource changed mode: for example, a job station, a test head, or the entire tester. Since no restrictions are placed on what value can be placed in this field, users are free to determine what modes will be monitored for utilization reporting and using what language. Typical values for the Type field include:

  ```
  tester        head
  station       site
  printer       pmu
  ```

  Any value is legal.

- The Identifier field can be any string that, when used with the Type name, uniquely identifies what has changed mode. For example, since some test systems do not have unique site numbers across heads, it is necessary to identify the site number in terms of the head. Site #4 on head #2 might therefore be identified by using the string **2.4**.

- The Mode field can contain any string (in any language). Typical Modes for a job station are:

  ```
  production     engineering
  maintenance    idle
  ```

(continued)

Examples:

This message indicates that test head 2 of tester zugspitze is now idle:

```
t_change_mode   zugspitze   9-jun-94_19:10:01   head   2   idle
```

This message is sent when tester ararat enters production mode:

```
t_change_mode   ararat   9-jun-94_19:10:01   tester   cadillac   production
```

This message might be sent when an engineer logs in to a station so that his time may be charged against the current project he is working on:

```
t_change_mode   hermon   13-jul-94_11:21:42   station   1   engineering
        "J. Lister -- debugging test program for I/O chip"
```

This message might be sent when the tester fails its checkers and is “down” awaiting maintenance:

```
t_change_mode  cadillac  13-jul-94_07:15:54  tester  ararat  down  "Checkers failed"
```

Equipment Failure Message (t_system_failure)

Function: To declare that a failure has occurred.

Syntax: **t_system_failure**

Source Node

Timestamp

Identifier

Reporter's Name

Comment

Example:

```
t_system_failure   kraken   26-NOV-91_20:30:00 999   smith   "Chan. 3 does not calibrate"
```

Equipment Restored Message (t_system_restore)

Function: To declare that a system has been partially or fully restored.

Syntax: **t_system_restore**

Source Node

Timestamp

Identifier

Reporter's Name

Option Name

Suboption Name

Comment

Notes:

- The Identifier field is used to tie the failure and restore messages together.
- Option Name is the name of the failed option. To indicate a system failure, use "system" in this field. If no fault is found, use "no fault" in this field.
- Suboption Name can be used as further identification for the failed option.

Example:

```
t_system_restore   kraken   26-NOV-91_21:30:00 999   smith   Computer   Mouse
        "Intermittent cable connection"
```

CPU Utilization Message (t_cpu_util)

Function: Used by a test system to indicate what percentage of time the tester computer CPU was active during the reporting period.

Syntax: **t_cpu_util**

Source Node

Timestamp

Time of Start of Utilization Period

Time of End of Utilization Period

Percent CPU Utilization

Notes:

- Start and end times use the format of the standard SEMF timestamp. See page 7.

Example:

```
t_cpu_util   etna   9-jun-94_08:01:00   9-jun-94_00:00:00   9-jun-94_08:01:00   76.7
```

Tester Utilization Message (t_tester_util)

Function: Used by a test system to indicate what percentage of time the tester spent testing devices. This information is reported without regard to whether it was also performing other activities in parallel.

Syntax: **t_tester_util**

Source Node

Timestamp

Time of Start of Utilization Period

Time of End of Utilization Period

Percent Time Testing Parts

Notes:

- Start and end times use the format of the standard SEMF timestamp. See page 7.
- Percent Time Testing Parts is defined as the percentage of time that the tester was actually testing parts. This is equivalent to the percentage of time that the “test in progress” light was on.

Example:

```
t_tester_util   everest   19-jul-94_11:11:23   19-jul-94_11:01:19   19-jul-94_11:11:19   82.3
```

Resource Utilization Message (t_resource_util)

Function: A generic message used for specifying the percentage of time the indicated resource was doing useful work. For example, what percentage of time was the printer actually printing? What percentage of time was a particular program running? What percentage of time was the pattern generator being used?

Syntax: **t_resource_util**

Source Node

Timestamp

Time of Start of Utilization Period

Time of End of Utilization Period

Type

Identifier

Utilization

Notes:

- Start and end times use the format of the standard SEMF timestamp. See page 7.
- The Type field indicates what kind of resource is being measured. For instance, was it a job station, a test head, or the entire tester. Since no restrictions are placed on what value can be placed in this field, users are free to determine what resources will be monitored for utilization reporting and using what language. Typical values for the Type field include:

```
head          station       handler
printer       pmu           prober
software      site          pattern_generator
```

  Any value is legal.

- The Identifier field can be any string that, when used with the Type name, uniquely identifies the resource being measured.

Example:

```
t_tester_util  everest  19-jul-94_11:11:23  19-jul-94_11:01:19  19-jul-94_11:11:19
        program  adart  5.6
```

Are You Alive Message (t_alive)

Function: Used by a network node to request that another node send status information. This information may be only an indication that the network partner is still alive on the network.

Syntax: **t_alive**

Source Node

Timestamp

Example:

```
t_alive   brocken   9-jun-94_08:01:23
```

Ask CPU Utilization Message (t_ask_cpu_util)

Function: Used by a network node to request that another node send its current CPU utilization as a percent used.

Syntax: **t_ask_cpu_util**

Source Node

Timestamp

Example:

```
t_ask_cpu_util   wachusett   9-jun-94_08:16:07
```

Send Text To Operator Message (t_opermsg)

Function: Used to send a text message to the operator of network node.

Syntax: **t_opermsg**

Source Node

Timestamp

Destination Station Name

Text Message

Notes:

- The word **ALL** may be used in the Destination Station Name field to indicate that the message is to be sent to all users on the indicated node.

Example:

```
t_opermsg   etna   9-jun-94_08:01:23   2   "System going down in 5 minutes"
```

Send Text To Event Log Message (t_event_log)

Function: Used by a network node to record a text message in the network event log.

Syntax: **t_event_log**

Source Node

Timestamp

Sender

Text Message

Example:

```
t_event_log   blanc   9-jun-94_08:01:23   Mary   "Lot D002534A missing 2 wafers"
```

General Reply Message (!t_reply)

Function: Used to send completion status information in response to a particular SEMF message. Reply messages are only sent if requested in the original SEMF message. The process initiating the original SEMF message is responsible for determining when to stop waiting for a reply.

Syntax: **!t_reply**

Source Node

Timestamp

Status

Return address

Notes:

- The Status parameter may be any arbitrary ASCII text string that conveys useful information between the sender and the receiver.
- The Return Address parameter contains the information necessary to direct the reply message to the process which is awaiting it. It is copied without change from the RSVP parameter in the original SEMF request message.

Example:

```
!t_reply   kebnekaise   9-jun-94_08:11:23   fail_17   @23@fe0@201e5@etna@inet@
```

Practical Restrictions

Because the SEMF is designed to be implemented under a number of different operating systems, it imposes few restrictions on what a message can contain. Nevertheless, a specific operating system may have requirements that make it impractical to send certain types of messages. You must therefore be aware of the practical limitations of the systems in your network before using the SEMF specification to design factory automation software. This section describes a number of practical restrictions as examples of the type of situation to avoid.

Tokens in an SEMF Message

While SEMF places no restriction on the number of tokens in an SEMF message, it is important to consider the restrictions of the programs that will be executed as a result of the message. For example, if the dispatched event handler is a VMS command file, you are allowed only eight tokens following the keyword. If it is a program written in a high-level language, you should check the restrictions on the language to make sure that it can handle the number of tokens you intend to pass. Some languages may provide tricks that will allow you to combine several pieces of information into a single token.

Length of the SEMF Message

The SEMF specification also places no limit on the number of characters in the message. However, in any given implementation of an Event Dispatcher, there will always be some practical limit. Check the documentation for the event dispatcher on the source and destination computer to make sure that the message size you want to send is allowed. Also make sure that the command line interpreter (or shell) on which the event program is to be executed can handle a message of the desired length.

Implementation Hints

The following are a few hints that can make it easier for test system software developers to provide the information necessary for effective use of event dispatching using the Standard Event Message Format.

Yield

In some testers, it is not possible for the tester executive to know what the current (or final) yield is without some help because it doesn't know which bins are good and which are bad for a given job plan. However, overall yield and yield alarms are very important to production managers. Therefore, it is important for the job plan language to provide a means to pass this information to the tester executive.

There are a number of alternatives for passing this information. One method, used by Teradyne's J937 memory tester, is to indicate, as a device is being binned, whether it passes or fails. Another is to provide a function call or set of flags by which the job plan can designate at the start of the job which are the passing bins. Either of these methods would allow the executive to keep track of yield in real time. A third method is to provide a function call in which the job plan passes the yield, which it calculated, to the central executive.

Handler and Prober Information

It is important for the job plan language to provide a means for the user to find out what the handler or prober is doing and to tell the executive about it. Such information as the current X/Y coordinates, jams, misalignments, and utilization are very important to test floor managers trying to keep production at a high level. The HIM, PIM, and HPIM routines from the J937 memory tester are useful for passing this kind of information.

General Test Floor Status

Among the pieces of status information that are useful to test floor managers in real time are the operator name, the name of the job plan, the job plan version, part type, and lot id. Some of these are best entered from the keyboard before the job is loaded, and some are better set by the job plan itself. In either case, it is necessary to provide a way for the central executive to find out this information, so it can be provided to the test floor monitor in status messages.

Main Menu

SEMF Specification

Index

Click on any page number.