8 Log Record Format

This chapter describes the format of the standard log records generated by the test system for use by Pushbutton Q-STATS. Although an in-depth understanding of log records is not necessary for general users of the test system, it is essential for those who wish to integrate Agilent board test data with non-Agilent (custom) applications.

Objectives

When you finish reading this chapter, you should be able to:

- Understand the structure and syntax of the log records

What are Log Records?

Test data is stored in files as a series of formatted records called log records. Each record consists of a character string which represents information. Because the records must be read and interpreted by programs, the characters in each record are arranged in precise formats. You must be familiar with these formats if you wish to interpret existing log records or generate your own from scratch.

NOTE Because log records are simply strings of ASCII characters, you can load a log record file into the BT-Basic workspace and examine it. Or, if you are familiar with the shell, you can examine log records with the `more` command or an editor such as vi. See Structure Of The Datalogging Files for the location of the log files.

About Log Record Prefixes

All standard log records generated by your test system contain a prefix that consists of the @ symbol followed by several descriptive characters that uniquely identify the type of record. For example, prefix @BLOCK indicates that the record describes a block, or group of tests. Although there can be any number of @BLOCK log records generated, only one type of log record can begin with an @BLOCK prefix.

The records are described in Table 8-1.

Table 8-1 Log record prefixes

Prefix	Purpose	Generated by
@A-CAP	capacitor test results	capacitor
@A-DIO	diode test results	diode
@A-FUS	fuse test results	fuse
@A-IND	inductor test results	inductor
@A-JUM	jumper test results	jumper
@A-MEA	measurement results	measure
@A-NFE	N-channel FET test results	nfetr
@A-NPN	NPN transistor test results	npn
@A-PFE	P-channel FET test results	pfetr
@A-PNP	PNP transistor test results	pnp
@A-POT	potentiometer test results	potentiometer
@A-RES	resistor test results	resistor
@A-SWI	switch test results	switch
@A-ZEN	zener test results	zener
@ALM	identify a real-time alarm	`alarm` program
@AID	identify board causing an alarm	`alarm` program
@ARRAY	digitizer results analysis	digitizer
		report analog
@BATCH	batch identifier	log board
@BLOCK	test block identifier	test
		test analog
@BS-CON	describe boundary-scan test	Boundary-Scan

Table 8-1 Log record prefixes (continued)

Prefix	Purpose	Generated by
@BS-O	list of open pins (boundary-scan)	Boundary-Scan
@BS-S	list of shorted pins (boundary-scan)	Boundary-Scan
@BTEST	describe board test	log board start
		log board end
@DPIN	list failing pins for one device	cont digital
		probe
		test
@D-PLD	PLD programming results	log is, log out, log failure, and other BT-Basic logging functions
@D-T	digital shorts test results	test
@INDICT	list potentially failing devices	log level is indictments
@LIM2	analog test high/low limits	current
		diode
		fuse
		jumper
		measure
		nfetr
		npn
		pfetr
		pnp
		switch
@LIM3	analog test nominal and tolerance limits	capacitor
		inductor
		potentiometer
		resistor
		zener
@NETV	network verification record	prsetup program
@NODE	list of nodes	probe
		probe failures

Table 8-1 Log record prefixes (continued)

Prefix	Purpose	Generated by
@PCHK	Polarity Check test results	test
@PIN	list of pins	pinsfailed
@PF	pins failure results	test
@PRB	probe failure results	probe
@RETEST	indicates log clear for retest	log clear for retest
@RPT	messages logged by `report`	report
@TJET	VTEP or TestJet results	test
@TS	shorts test results (destination)	test shorts
@TS-D	list of shorted nodes	test shorts
@TS-O	list of open nodes	test shorts
@TS-P	list of phantoms	test shorts
@TS-S	shorts test results (source)	test shorts

Note that several groups of prefixes exist. For example, records beginning with `@A-` are generated by analog test statements, records beginning with `@D-` are generated by digital test statements, and so on. If you are generating custom log records, you may want to create your own prefix groups.

NOTE To keep log records to a manageable size, avoid making custom prefixes excessively long.

Hierarchy of Log Records

This section shows how groups of log records are arranged to form log data files. To keep the examples simple, each record is represented by its prefix and not by the actual data it contains. The internal structure of each log record is explained later in Interpreting the Log Records on page 8-8.

Log records are arranged in a hierarchy of records and subrecords, where a subrecord is simply an additional record which further describes whatever precedes it. For example, a group of log records (presumably a log data file) might begin with an `@BATCH` record containing information to identify a batch of boards that were tested:

```
@BATCH
```

And the `@BATCH` record might be followed by an `@BTEST` record to identify the testing of an individual board:

```
@BATCH
@BTEST
```

Because the `@BTEST` record further describes the `@BATCH` record by identifying a unique board within the batch, it is a subrecord of, or subordinate to, the `@BATCH` record. The `@BATCH` record by itself is incomplete; it becomes complete only when followed by an `@BTEST` record.

Suppose the board described in the `@BTEST` subrecord had a resistor test performed on it (by a `resistor` statement). The results of the resistor test appear in an `@A-RES` record (which is preceded by an `@BLOCK` record to identify the beginning of a block of tests), like this:

```
@BATCH
@BTEST
@BLOCK
@A-RES
```

Thus, the `@A-RES` record appears as a subrecord of an `@BLOCK` record, which is a subrecord of an `@BTEST` record, which is a subrecord of an `@BATCH` record.

To make the structure of records and subrecords more obvious, we will begin indenting the examples, like this:

```
@BATCH
   @BTEST
      @BLOCK
         @A-RES
```

This way of presenting examples clearly shows the hierarchy of records and subrecords. Note that these indents are for illustration purposes only; they do *not* appear within actual log data files.

Suppose we test two more boards, each with one resistor test. The log data file now looks like this:

```
@BATCH
   @BTEST
      @BLOCK
         @A-RES
   @BTEST
      @BLOCK
         @A-RES
   @BTEST
      @BLOCK
         @A-RES
```

Completing the current batch of boards and beginning a new one gives us:

```
@BATCH
   @BTEST
      @BLOCK
         @A-RES
   @BTEST
      @BLOCK
         @A-RES
   @BTEST
      @BLOCK
         @A-RES
@BATCH
   @BTEST
      @BLOCK
         @A-RES
```

In a similar fashion, other records and subrecords can be assembled into files of log data describing how boards were tested.

Interpreting the Log Records

Up until now, all of the examples have shown only the record prefixes, and not the test data each record contains. This section describes the format of data as it appears within log records and then expands upon the preceding section by showing how actual log records are grouped into log data files.

NOTE Although the following examples illustrate the structure of actual log records, the samples shown are not necessarily complete. Refer to the descriptions of the individual log records for complete details.

Each log record is enclosed in braces. The record begins with { immediately followed by the prefix, and ends with }. For example, the @A-CAP record, which describes a capacitor test, looks like this when simplified:

```
{@A-CAP. . . data fields . . .}
```

The prefix is followed by one or more variable length data fields containing descriptive information. A vertical bar, |, is used as a separator between fields. Adding the names of the fields to the example above gives us:

```
{@A-CAP|test status|measured value|subtest designator}
```

NOTE Look in the descriptions of the individual log records for the names of the fields associated with each log record.

If we replace the names of the fields with typical data, an @A-CAP record might look like this:

```
{@A-CAP|1|1.246700E+01|C1}
```

If a field is optional, the separator character must still appear as a place holder. For example, if the @A-CAP record above had no value assigned to the measured value field, it would look like this:

```
{@A-CAP|1||C1}
```

Notice that the vertical bars still reserve a space between the contents of the test status and subtest designator fields for the empty measured value field.

Each field must contain a certain type of information. The log record descriptions denote these beneath TYPE as shown in Table 8-2.

Table 8-2 Log record types

Type	Description
bool	a boolean, which can have either a `true` value (1 or Y) or a `false` value (0 or N)
fp	floating point number, which is an integer optionally followed by a decimal point and any number of consecutive digits, all optionally followed by the letter E (upper- or lowercase) and an integer
int	integer number, which is an optional sign character (+ or -) followed by one or more consecutive digits
str	string value, which is any number of characters of any type (unless otherwise noted)

If left empty, most fields default to some predefined value. The log record descriptions denote these beneath `DEFAULT` as:

- 0, 1, etc. a default numeric value
- "" a null string

Fields for which a default value would be meaningless - that is, fields in which a reported value is mandatory - have a question mark (?) beneath `DEFAULT`.

Some log records contain a data list. In this case, the initial separator character (to indicate that a list follows) is a backslash, \, and individual items within the list are separated by a vertical bar. For example, the `@NODE` log record contains a list of node identifiers:

```
{@NODE\node list|item 1|item 2| . . etc.}
```

Substituting values in place of the field names gives us:

```
{@NODE\2|Node53|+5Volts}
```

The first entry in a list tells how many items appear in the list. Thus, the 2 following the \ indicates there are two items in the list: `Node53` and `+5Volts`.

Some log records are followed by one or more subrecords containing additional information. For example, the `@PF` record can be followed by an `@PIN` record which contains a list of pins:

```
{@PF|2
   {@PIN\2|11434|22216}
}
```

In other words, the `@PF` record is further described by the `@PIN` record that follows it.

Notice the hierarchy of braces - the @PIN record has its own pair of beginning and ending braces ({ and }) within the @PF record's overall pair of braces. Also notice that the closing brace for the @PF record was dropped onto a new line, and the @PIN record was indented, to make the structure of the overall record more obvious. Indents and new lines are shown here for illustration purposes only; they do not necessarily appear within actual log data files.

If you examine actual log data produced by a Medalist ICT system, you will see that in some cases subrecords appear on the same line as the records with which they are associated, and in other cases they do not. The important thing to keep in mind is that log records must appear in the correct order and that they must be accurately defined by braces.

Without the indents and new lines, the actual log record looks like this:

```
{@PF|2{@PIN\2|11434|22216}}
```

This same hierarchy of braces and indents also applies to more complex, nested groups of log records. Consider the following example:

```
{@TS|1|2|1|1|
   {@TS-S|2|0|Node12
      {@TS-D|Node25|1.678850E+00}
      {@TS-D|Node26|2.543211E+00}
   }
   {@TS-O|Node43|Node14|-1.500000E+00}
   {@TS-S|0|1|Node38
      {@TS-P|-1.243853E+02}
   }
}
```

In this case an overall @TS log record contains three subrecords, one each to report shorts, opens, and phantoms associated with the shorts test reported by the entire @TS log record. Two of the @TS record's subrecords contain subrecords which further describe them. Notice the pairing of braces; for each opening brace, there is a closing brace.

So far we have mentioned four special characters which can appear in log records: {, }, |, and \. But suppose that one or more of these characters must appear within a data field and not be interpreted as data field delimiters. Another special character, the tilde (~), is used to identify a data field that should be interpreted literally, that is, a literal field. Unlike a normal field, which is variable length, this literal field must have its length precisely defined. For example:

```
~12|!$%(test)|!|
```

The actual data in this field is !$%(test)|!|. The ~ identifies the field as a special case, the 12 denotes the field's length as twelve characters, and the | immediately following the 12 delimits the length specification from the data.

If this field appeared in a log record, it might look like this:

```
{@RPT~12|!$%(test)|!|}
```

Notice that the ~ at the beginning of the literal field replaces the usual | delimiter; do not use both delimiters.

If you generate custom log records containing the line feed character, an ASCII 10 (CTRL-J), you must put the line feed character in a literal field; if you do not, it will be misinterpreted as an optional delimiter between a record and a subrecord.

If you need to manipulate log records with BT-Basic, you should use free-field formatting to input a record's individual fields into variables.

A Note About Truncated Records

If you are using custom datalogging routines, be aware that the presence of a truncation character, an ASCII 4 (CTRL-D), in the data will be interpreted to mean that the log data was unexpectedly interrupted and that the current log record has ended.

NOTE The exception to this is that the truncation character can appear in a literal field denoted with a tilde (~); this does not indicate a truncated record.

The data preceding the truncation character is considered to be valid, and any subsequent log records will be considered normal provided they do not contain the truncation character.

A Note About Process Steps

In addition to type and serial number, boards can be identified by a process step. The process step is simply at which step in the manufacturing process - for example, in-circuit or functional - the board is being tested. The `@BATCH` log record contains a process step data field in which you can place a brief code to identify at which step in the test process a board was tested.

Note that Pushbutton Q-STATS does not make use of process steps. However, if you are generating custom log records and ever expect to use the optional Q-STATS II quality management software (which does use process steps), you may want to begin acquiring process step information now.

A Note About User-Defined Log Records

The `log` and `log using` statements allow you to create user-defined log records in free-field and imaged formats, respectively. User-defined log records can emulate existing log records, or they can be created from scratch for custom applications (which require custom routines to process the non-standard log records).

How Log Records are Formatted

The next two topics summarize how log records are formatted. This information will be useful if you are creating your own custom log records.

- Special Characters
- Rules of Formatting

Special Characters

The characters shown in Table 8-3 have special significance - that is, they are interpreted to mean something other than their normal ASCII value - when they appear in a log record.

Table 8-3 Special characters

Character	Description
{	Begins a log record
}	Ends a log record
|	Begins a normal data field
~	Begins a literal field
\	Begins a list of fields
ASCII 4 (CTRL-D)	Identifies a truncated (incomplete) log record
ASCII 10 (CTRL-J)	Line feed character (new line). Denotes the end of the data fields for the current record, and should only be followed by an end of record character (}) or beginning of a new record or subrecord ({).

If you use custom log records or custom datalogging routines, be sure that none of these characters appear in normal data unless they are within a literal field.

Rules of Formatting

The following list contains the formatting rules used to build log records (or subrecords).

- Each log record begins with { followed by a series of printable characters that uniquely identify the log record type. The convention for log record types supplied by Agilent is that they begin with @. No other log records should begin with this character.
- The record type is followed by one or more data fields.
- Each data field begins with one of the following characters:

  | - Begins a normal data field, which must not contain special characters unless they appear within a literal field. A normal data field is ended by the appearance of the next special character. If you are creating custom log records: Since a special character ends a data field, every data field should be followed by some special character to terminate it.

  \ - Begins a data field that contains a list, which must not contain any special characters unless they appear within a literal field. This character is followed by:

  - A number showing how many items are in the list
  - The items (data) in the list. Each item is itself a data field, and usually begins with |.

    ~ - Begins a literal field, which is the only way to include special characters inside a log record and not have them interpreted as special. This character is followed by:

  - A number showing how many characters are in the field
  - A single |
  - The characters which are to be interpreted literally
- Each log record ends with }
- Some record types can contain nested subrecords. A record becomes a subrecord when it occurs after the preceding record's data fields but before the final } for the preceding record.

Descriptions of the Log Records

This section describes the log records.

Record @A-CAP: capacitor

This record describes a capacitor test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-4 @A-CAP

Format: {@A-CAP | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `capacitor` statement

- Generated by: `capacitor` statement
- Subrecords: `@LIM3` is generated when logging limits information. Contains nominal value and high/low limits for test.

Example 8-1

```
{@A-CAP|1|1.246700E+01}
```

Record @A-DIO: diode

This record describes a diode test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-5 @A-DIO

Format: {@A-DIO | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `diode` statement.

- Generated by: `diode` statement
- Subrecords: `@LIM2` is generated when logging limits information. Contains high/low limits for test.

Example 8-2

```
{@A-DIO|1|1.246700E+01}
```

Record @A-FUS: fuse

This record describes a fuse test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-6 @A-FUS

Format: {@A-FUS | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `fuse` statement.

- Subrecords: `@LIM2` is generated when logging limits information. Contains high/low limits for test.

Example 8-3

```
{@A-FUS|1|1.246700E+01}
```

Record @A-IND: inductor

This record describes an inductor test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-7 `@A-IND`

Format: {@A-IND | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `inductor` statement.

- Generated by: `inductor` statement
- Subrecords: `@LIM3` is generated when logging limits information. Contains nominal value and high/low limits for test.

Example 8-4

```
{@A-IND|1|1.246700E+01}
```

Record @A-JUM: jumper

This record describes a jumper test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-8 `@A-JUM`

Format: {@A-JUM | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `jumper` statement.

- Generated by: `jumper` statement
- Subrecords: `@LIM2` is generated when logging limits information. Contains high/low limits for test.

Example 8-5

```
{@A-JUM|1|1.246700E+01}
```

Record @A-MEA: measure

This record describes a measurement. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-9 `@A-MEA`

Format: {@A-MEA | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 7 = failed 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the name of the subtest block in which the measurement occurred.

- Generated by: `measure` statement
- Subrecords: `@LIM2` is generated when logging limits information. Contains high/low limits for test.

Example 8-6

```
@A-MEA|7|-3.654285E-05|N-FET_ON_OFF{@LIM2|+5.000000E+00|-5.000000E-01}
```

Record @A-NFE: nfetr

This record describes an N-channel FET test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-10 `@A-NFE`

Format: {@A-NFE | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `nfetr` statement.

- Generated by: `nfetr` statement
- Subrecords: `@LIM2` is generated when logging limits information. Contains high/low limits for test.

Example 8-7

```
{@A-NFE|1|1.246700E+01}
```

Record @A-NPN: npn

This record describes an NPN transistor test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-11 `@A-NPN`

Format: {@A-NPN | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `npn` statement.

- Generated by: `npn` statement
- Subrecords: `@LIM2` is generated when logging limits information. Contains high/low limits for test.

Example 8-8

```
{@A-NPN|1|1.246700E+01}
```

Record @A-PFE: pfetr

This record describes a P-channel FET test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-12 `@A-PFE`

Format: {@A-PFE | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `pfetr` statement.

- Generated by: `pfetr` statement
- Subrecords: `@LIM2` is generated when logging limits information. Contains high/low limits for test.

Example 8-9

```
{@A-PFE|1|1.246700E+01}
```

Record @A-PNP: pnp

This record describes a PNP transistor test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-13 `@A-PNP`

Format: {@A-PNP | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `pnp` statement.

- Generated by: `pnp` statement
- Subrecords: `@LIM2` is generated when logging limits information. Contains high/low limits for test.

Example 8-10

```
{@A-PNP|1|1.246700E+01}
```

Record @A-POT: potentiometer

This record describes a potentiometer test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-14 `@A-POT`

Format: {@A-POT | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `potentiometer` statement.

- Generated by: `potentiometer` statement
- Subrecords: `@LIM3` is generated when logging limits information. Contains nominal value and high/low limits for test.

Example 8-11

```
{@A-POT|1|1.246700E+01}
```

Record @A-RES: resistor

This record describes a resistor test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-15 `@A-RES`

Format: {@A-RES | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `resistor` statement.

- Generated by: `resistor` statement
- Subrecords: `@LIM3` is generated when logging limits information. Contains nominal value and high/low limits for test.

Example 8-12

```
{@A-RES|1|1.246700E+01}
```

Record @A-SWI: switch

This record describes a switch test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-16 `@A-SWI`

Format: {@A-SWI | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `switch` statement.

- Generated by: `switch` statement
- Subrecords: `@LIM2` is generated when logging limits information. Contains high/low limits for test.

Example 8-13

```
{@A-SWI|1|1.246700E+01}
```

Record @A-ZEN: zener

This record describes a Zener diode test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-17 `@A-ZEN`

Format: {@A-ZEN | test status | measured value | subtest designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed 2 = failed (compliance limit) 3 = failed (detector timeout) 11 = aborted by operator
measured value	fp	0	The actual value measured.
subtest designator	str	""	Comes from the optional `designator` parameter of the `zener` statement.

- Generated by: `zener` statement
- Subrecords: `@LIM3` is generated when logging limits information. Contains nominal value and high/low limits for test.

Example 8-14

```
{@A-ZEN|1|1.246700E+01}
```

Record @AID: identify board causing real-time alarm

This record appears as a subrecord of the `@ALM` record; it identifies the board that caused the real-time alarm described in the `@ALM` record.

Table 8-18 `@AID`

Format: {@AID | time detected | serial number}			
Field	**Type**	**Default**	**Comments**
datetime detected	str	""	Date and time of alarm in YYMMDDHHMMSS format (datetime$).
subtest designator	str	""	The unique serial number of the board that caused the real-time alarm.

- Generated by: `alarm` program

Example 8-15

```
{@AID|890615094418|12306743}
```

Record @ALM: identify real-time alarm

This record describes a specific real-time alarm. It is followed by an `@AID` subrecord that identifies which board caused the real-time alarm.

Table 8-19 `@ALM`

Format: {@ALM | alarm type | alarm status | time detected | board type | board type rev | alarm limit | detected value | controller | testhead number}			
Field	**Type**	**Default**	**Comments**
alarm type	int	1	1 = consecutive failure; 2 = sample yield; 3 = overall yield
alarm status	bool	0	0 = off, 1 = on
datetime detected	str	""	Date and time of alarm in YYMMDDHHMMSS format (datetime$).
board type	str	""	The board type associated with the alarm.
board type rev	str	""	The revision of the board type associated with the alarm.
alarm limit	int	?	The alarm limit (threshold) that was exceeded.
detected value	int	?	The detected value that exceeded the alarm limit.
controller	str	""	Which controller the alarm occurred on.
testhead number	int	1	(th$) a positive integer.

- Generated by: `alarm` program
- Subrecords: `@AID` Logged for any real-time alarm that is detected; identifies the board that caused the real-time alarm.

Example 8-16

```
{@ALM|1|1|890516145512|proc_bd|2|10|15|alpha|1}
```

Record @ARRAY: digitizer results analysis

This record describes the results of a `measure` statement for the digitizer or the results of an array-format `report analog` statement.

Table 8-20 `@ARRAY`

Format: {@ARRAY | subtest designator | status | failure count | samples}			
Field	**Type**	**Default**	**Comments**
subtest designator	str	""	Comes from the digitizer or from the optional `designator` parameter of the `report analog` statement.
status	int	0	0 = pass 1 = fail 7 = error occurred
failure count	int	0	The number of failures.
samples	int	1024	The number of samples tested.

- Generated by: `digitizer`, `report analog` statement

Example 8-17

```
{@ARRAY|""|1|5|1024}
```

Record @BATCH: identify a batch of boards

This record identifies a unique batch of boards; that is, a number of boards which are treated as a group for test purposes.

Table 8-21 @BATCH

Format: {@BATCH | UUT type | UUT type rev | fixture id | testhead number | testhead type | process step | batch id | operator id | controller | testplan id | testplan rev | parent panel type | parent panel type rev}			
Field	**Type**	**Default**	**Comments**
UUT type	str	""	The type of unit under test, not including the revision. This can be a board type or a panel type.
UUT type rev	str	""	The board type revision or panel type revision.
fixture id	int	0	The decimal autofile code.
testhead number	int	1	(th$) a positive integer.
testhead type	str	""	Which type of testhead. (This is currently not used.)
process step	str	""	Which step in the manufacturing process.
batch id	str	""	ID of current batch of boards.
operator id	str	""	Which operator.
controller	str	""	Which controller.
testplan id	str	""	Name of testplan.
testplan rev	str	""	Revision of testplan.
parent panel type	str	""	Type of the panel that contains this board.
parent panel type rev	str	""	Revision of the type of panel that contains this board.
version label	str	""	Multiple Board version.

- Generated by: `log board` statement
- Subrecords:
  - `@BTEST` Generated for each board within a batch. When one log data file is generated per board tested, the file has only one `@BATCH` record which in turn has only one `@BTEST` subrecord.
  - `other` The `log` and `log using` statements can be used to generate other subrecords.

Example 8-18 Example of @BATCH log record

```
{@BATCH|998457-146|0|2550|1||btest|891131172938|pete|achilles|MaxWellBT|7|A_panel|2}
```

Record @BLOCK: identify a test block

This record identifies a named block of analog, digital, or mixed test statements.

Table 8-22 `@BLOCK`

Format: {@BLOCK | block designator | block status}			
Field	**Type**	**Default**	**Comments**
block designator	str	""	Name of this test block.
block status	int	0	0 = all statements passed. If a statement failed, the status of the failing test is shown.

- Generated by: `test`, `test analog` statements
- Subrecords: Subrecords are shown in Table 8-23.

Example 8-19

```
{@BLOCK|R12|1
   {@A-RES||1|1.006789E+01}
}
```

Table 8-23 `BLOCK` subrecords

Subrecord	Generated by	Subrecord	Generated by
@A-CAP	`capacitor` statement	**@A-PNP**	`pnp` statement
@A-DIO	`diode` statement	**@A-POT**	`potentiometer` statement
@A-FUS	`fuse` statement	**@A-RES**	`resistor` statement
@A-IND	`inductor` statement	**@A-SWI**	`switch` statement
@A-JUM	`jumper` statement	**@A-ZEN**	`zener` statement
@A-MEA	`measure` statement	**@D-T**	`test` statement
@A-NFE	`nfetr` statement	**@S-PROC**	`test` statement
@A-NPN	`npn` statement	**@TJET**	`test` statement
@A-PFE	`pfetr` statement		

Record @BS-CON: boundary-scan test

This record describes a boundary-scan test.

Table 8-24 `@BS-CON`

Format: {@BS-CON | test designator | status | shorts count | opens count}			
Field	**Type**	**Default**	**Comments**
test designator	str	""	Designator for a boundary-scan connect or interconnect test.
status	int	0	0 = pass 1 = fail 7 = chain failure
shorts count	int	0	How many shorts occurred during the boundary-scan test.
opens count	int	0	How many opens occurred during the boundary-scan test.

- Generated by: Boundary-Scan Software
- Subrecords:
  - `@BS-O is` generated when `opens count` is greater than zero and the log level is not set to `without pins`.
  - `@BS-S` is generated when `shorts count` is greater than zero and the log level is not set to `without pins`.

Example 8-20

```
{@BS-CON|9c_connect|1|0|2
   {@BS-O|9C|43}
   {@BS-O|9C|41|9C|58}
}
{@BS-CON|27c_connect|1|0|1
   {@BS-S|S{@NODE\2|179|112}}
}
```

Record @BS-O: list of open pins

This subrecord follows a `@BS-CON` log record and further describes that record by listing open pins that were found during a failing boundary-scan test.

Table 8-25 `@BS-O`

Format: {@BS-O | first device | first pin | second device | second pin}			
Field	**Type**	**Default**	**Comments**
first device name	str	""	Name of the first device.
first device pin	int	1	Number of open pin on the first device.
second device name	str	""	(optional) Name of the second device.
second device pin	int	1	(optional) Number of open pin on the second device.

- Generated by: Boundary-Scan Software

Example 8-21

```
{@BS-O|9C|43}
{@BS-O|9C|41|9C|58}
```

Record @BS-S: list of shorted pins

This subrecord follows an `@BS-CON` log record and further describes that record by listing a shorted pin that was found during a failing boundary-scan test.

Table 8-26 `@BS-S`

Format: {@BS-S | cause | node list}			
Field	**Type**	**Default**	**Comments**
cause	str	S	S = known short U = unknown short 0 = stuck at 0 1 = stuck at
node list	str	""	A list of indicted nodes.

- Generated by: Boundary-Scan Software

Example 8-22

```
{@BS-S|S{@NODE\2|179|112}}
```

Record @BTEST: describe a board test

This record describes the overall testing of a single board. It is followed by subrecords which report the results of individual tests performed on the board.

Table 8-27 `@BTEST`

Format: {@BTEST | board id | test status | start datetime | duration | multiple test | log level | log set | learning | known good | end datetime | status qualifier | board number | parent panel id}

Field	Type	Default	Comments
board id	str	""	Serial number of this board.
test status	int	0	0 = passed 1 = uncategorized failure 2 = failed pin test 3 = failed in `learn` mode 4 = failed shorts test 5 = (reserved) 6 = failed analog test 7 = failed power supply test 8 = failed digital or boundary scan test 9 = failed functional test 10 = failed pre-shorts test 11 = failed in board handler
			12 = failed barcode 13 = X'd out (board on panel not tested or missing) : maintained, but not logged. 14 = failed in VTEP or TestJet 15 = failed in polarity check 16 = failed in ConnectCheck (Mux system only) 17 = failed in analog cluster test 18-79 = reserved 80 = runtime error 81 = aborted (STOP) 82 = aborted (BREAK) 83-89 = reserved 90-99 = user-definable
start datetime	int	0	Start of board test in YYMMDDHHMMSS format (datetime$).
duration	int	0	Test duration in seconds.
multiple test	bool	0	Is this the same as the previous board? 0 = NO, 1 = YES
log level	str	""	Current value of `lli$`.
log set	int	0	Current default logging set. (This is not currently used.)
learning	bool	0	Is learning on? N = NO, Y = YES

Table 8-27 `@BTEST (continued)`

Format: {@BTEST | board id | test status | start datetime | duration | multiple test | log level | log set | learning | known good | end datetime | status qualifier | board number | parent panel id}			
Field	**Type**	**Default**	**Comments**
known good	bool	0	Is this a known good board? N = NO, Y = YES
end datetime	int	0	End of board test in YYMMDDHHMMSS format (datetime$).
status qualifier	str	""	Comment string from `log board end`.
board number	int	1	The number of the board being tested, as determined by a `board number is` statement. When not using multiple log buffers (`board number is *`), this value is reported as 1 even though it is actually zero.
parent panel id	str	""	Value derived from the optional `<ParentPanelId>` parameter of a `log board start` statement.

NOTE The test status constants (above) are defined in the test plan, and are used by statistical analysis and quality control programs such as PushButton QStats and other packages. These tools respond primarily to the category of `PASS/FAIL/BOGUS`, though they do sometimes recognize that the value 2 is special. Pins test failure can be a bad board, or a problem with the fixture.

- Generated by:
  - `log board start` statement begins this record.
  - `log board end` statement ends this record.

Pushbutton Q-STATS and Q-STATS II interpret a test status of 1 through 10 as a failing board, while values 11 through 99 are considered bogus; that is, they are neither passing nor failing.

Example 8-23

```
{@BTEST|99538-135|8|891131172855|43|0|failures||n|n|891131172938|4|99538-130
         . . .
      {test results subrecords}
         . . .
   }
```

Table 8-28 `BTEST` subrecords

Subrecord	Generated by
@BLOCK	`test` and `test analog` statements
@BLINE	Baseline status information (see Version Tracking with Enhanced Log Records).
@BS-CON	Boundary-Scan connect and interconnect tests
@D-LOG	`log digital` statement
@D-T	`test` statement
@PF	`pinsfailed` function
@PRB	`probe` or `probe failures` statements
@RETEST	`log clear for retest` statement
@RPT	any of the `report` statements: `report (ANALOG)`, `report (BT-Basic)`, or `report (SHORTS)`
@TS	`test shorts` statement
other	The `log` and `log using` statements can be used to generate other subrecords

Record @CCHK: Connect Check Test

This record that describes the results of Connect Check tests on a Mux system.

Table 8-29 `@CCHK`

Format: {@CCHK | test status | pin count | test designator}			
Field	**Type**	**Default**	**Comments**
test status	int	00	00 = pass 01 = fail 07 = fatal error
pin count	int	0000	number of failing pins
device designator	str	""	

- Generated by: `test` statement
- Subrecords: `@DPIN` is generated when logging pins to list the failing device pins.

Example 8-24

```
{@CCHK|01|0008|u34}
```

Record @DPIN: list of device pins for a single device

This record contains a list of device pins for a single device.

Table 8-30 `@DPIN`

Format: {@DPIN | device name | node pin list} or {@DPIN | device name | node pin list | thru devnode list} with DriveThru			
Field	**Type**	**Default**	**Comments**
device name	str	""	Name of device, or test designator when device name is not available.
node pin list	str or int	""/0	List of items. Items alternate to give two separate items of information about each pin. • The odd items are the `node id` strings. • Even items are the optional `device pin` strings to indicate which pins of this device are connected to the node id's. Device pins should be included whenever possible. The number of list items will always be even, that is, twice the number of device pins.
thru devnode list	str or int	""	List of DriveThru devices and nodes, ordered in pairs. Two items alternate information about each driven through device and node. • The odd items are the `node id` strings. • Even items are the `device id` strings to indicate which device is connected to the node id's. The number of list items will always be even, that is, twice the number of nodes driven through.

- Generated by:
  - `test` statement

    Tests for digital devices produce an `@DPIN` record as a subrecord for `@D-T` (digital powered test) and for `@TJET` (VTEP or TestJet test) whenever pins are being logged.
  - `probe` statement, `probe failures` statement

    These statements may produce `@DPIN` records, but as subrecords of the `@PRB` record when logging pins.

Example 8-25

```
{@DPIN|U12\4|Node17|8|GND|3}                ! No DriveThru
Node {@DPIN|U6809\6|TCLK|18|U6809-12|12|BDRV|18}
                                            ! With DriveThru Node
```

Record @D-PLD: results of PLD programming - success or failure

Data logging for PLD ISP is supported and controlled similarly to that of other test data types. Log record control is enabled via the BT-Basic logging functions, such as “log is”, “log out”, and “log failure”.

Log records produced for a PLD ISP test type are grouped into a digital test block record (@BLOCK). Every `play` statement in the digital test generates a sub-record within the block that describes information and the outcome of the action performed. Each block record also contains a digital sub-record describing the outcome of the digital test portion of the test. Each D-PLD record may contain NOTE and EXPRT sub-records.

A record is produced for a successful run with `log level all` or a failing execution with `log level is failures`. Otherwise no log is produced with the possible exception of using the `log devices` mechanism.

Table 8-31 `@D-PLD`

Format: {@D-PLD | <Filename> | <Action> | <Action return code> | <Result message string> | <Player program counter> | }

Field	Type	Default	Comments
Filename	str	“ “	Identifies the name of the compiled program file used by the test.
Action	str	“ “	Identifies the STAPL source action taken by the play statement. SVF and Jam files use a default name because procedural actions are not supported in these file types.
Action return code	int	0	Specifies the return code as a result of running the action. A zero (or empty parameter) designates a passing condition.
Result message string	str	“ “	Describes the error condition code.
Player program counter	int	0	Indicates the JBC opcode index where the failure occurred.

D-PLD may have one or more @EXPRT subrecords, a shown in Table 8-32. These are Generated by the use of STAPL export commands in the programming file.

Table 8-32 @EXPRT

Format: {@EXPRT | <Key> | <Field> }			
Field	**Type**	**Default**	**Comments**
Key	str	" "	Value that identifies the name of the variable exported.
Field	int	0	Represents the value of the variable when exported.

D-PLD may also have one or more @NOTE subrecords as shown in Table 8-33. This record is generated by the use of note header commands within the STAPL source file.

Table 8-33 @NOTE

Format: {@NOTE | <Note name> | <Note string> }			
Field	**Type**	**Default**	**Comments**
Note name	str	" "	Identifies a keyword name for the note field returned.
Note string	str	" "	Represents the contents of the note field returned.

- Generated by: `play` statement
- Subrecords: @EXPRT and @NOTE

The following example shows a record produced from a STAPL read ID code test program:

Example 8-26

```
{@BLOCK|d3_18v04|00
   {@D-PLD|digital/idtest.jam.jbc|read_idcode||||
      {@EXPRT|Expected is: |1BBBB44444444445555555AAAA4321}
      {@EXPRT|Got >> nd  : |01BBB44444444445555555AAAA4321}
      {@NOTE|CREATOR|Altera Chain Interrogation Version 2.02--Debug pipe mode}
      {@NOTE|DATE|2001/04/30}
      {@NOTE|ALG_VERSION|1}
      {@NOTE|STAPL_VERSION|JESD71}
      {@NOTE|MAX_FREQ|10000000}
   }
   {@D-T|0|384||0|d3_18v04}
}
```

Record @D-T: test digital

This record describes the results of a digital test. When pin logging is in effect, it is followed by a subrecord containing additional information.

Table 8-34 @D-T

Format: {@D-T | test status | test substatus | failing vector number | pin count | test designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed 1 = failed (see `test substatus` for reason) 5 =CRC related failure 7 = fatal error (test did not complete) 8 = pre or post chain integrity failure. This happens when the failing cell cannot be found in the map (.X) file information. The map file contains the information for the cells that are used in the test and will point to a device, pin, and node when that cell is failing.
test substatus	int	0	Decimal equivalent of a 6-bit binary value; a bit is set to 1 if the event occurred; to 0 if it did not occur. The bit meanings (lsb to msb) are: • **bit 0** fail • **bit 1** SAFEGUARD timeout • **bit 2** hardware error • **bit 3** pause • **bit 4** halt • **bit 5** overvoltage
failing vector number	int	0	Which vector failed.
pin count	int	0	How many pins failed.
test designator	str	""	Designator (name) of test.

- Generated by: `test` statement
- Subrecords: `@DPIN` is generated when logging pins to list the failing device pins.

Example 8-27

```
{@D-T|1|1|39|3|U18}
```

NOTE Flash ISP tests are logged like a normal digital test.

Record @INDICT: indict record

This record is used with DriveThru. This record contains a list of devices indicted by the respective test as either a potential failure or a known failure, and is generated when the `log level is` statement is set to `indictments` (or some higher level). When using DriveThru, this record is nested as a subrecord of `CPROBE` (`@TJET`).

Table 8-35 @INDICT

Format: {@INDICT | technique \ device list | est resistance | est capacitance | est induct | est model}			
Field	**Type**	**Default**	**Comments**
technique	str	""	`DT` to indicate DriveThru.
device list	str		The device list shows the reference designators for the indicted devices.The record can contain multiple device names (separated by a colon). For more information see the examples below, and refer to Interpreting the Log Records on page 8-8.
est resistance	fp		Not used for DriveThru.
est capacitance	fp		Not used for DriveThru.
est inductance	fp		Not used for DriveThru.
est model	str	""	Not used for DriveThru.

- Generated by: `test` statement

Example 8-28

```
{@INDICT|DT\1|r12}
{@INDICT|DT\3|rp6:r2|c412|r22}
```

Record @LIM2: high & low limits of analog test

This record contains the high and low limits for an analog test. When limits are being logged, it is followed by a subrecord containing additional information.

Table 8-36 `@LIM2`

Format: {@LIM2|high limit|low limit}			
Field	**Type**	**Default**	**Comments**
high limit	fp	0	Upper limit for allowable range.
low limit	fp	0	Lower limit for allowable range.

- Generated by:
  - `diode` statement
  - `fuse` statement
  - `jumper` statement
  - `measure` statement
  - `nfetr` statement
  - `npn` statement
  - `pfetr` statement
  - `pnp` statement
  - `switch` statement

Example 8-29

```
{@LIM2|8.666667E+01|2.000000E+00}
```

Record @LIM3: nominal high/low

This record is logged by statements which optionally log a component's nominal value and its high and low value limits. The limits are calculated from the nominal value, the plus and minus tolerance limits, and any margin in effect from a `tolerance margin` statement. The record is not generated if the log level was specified as `without nhls`.

Table 8-37 `@LIM3`

Format: {@LIM3|nominal value|high limit|minus tolerance}			
Field	**Type**	**Default**	**Comments**
nominal value	fp	0	The nominal value.
high limit	fp	0	Upper limit: The sum of the nominal value plus the allowed positive deviation from the nominal value.
low limit	fp	0	Lower limit: The sum of the nominal value plus the allowed negative deviation from the nominal value.

- Generated by:
  - `capacitor` statement
  - `inductor` statement
  - `potentiometer` statement
  - `resistor` statement
  - `zener` statement

Example 8-30

```
{@LIM3|22|1.500000E+00|2.000000E+00}
```

Record @NETV: network verification record

This record is used to verify the integrity of the network link to whichever system is specified.

Table 8-38 `@NETV`

Format: {@NETV | datetime | test system | repair system | source}			
Field	**Type**	**Default**	**Comments**
datetime	str	0	Date and time of verification request in YYMMDDHHMMSS format (datetime$).
test system	str	""	Identifier of test system.
repair system	str	""	Identifier of repair system.
source	bool	0	This field contains 0 when generated by `prsetup`, and changes to a 1 when the record is processed by `alarm`.

- Generated by: `prsetup` program

Example 8-31

```
{@NETV|890530102019|alpha|beta|1}
```

Record @NODE: list of nodes

This record contains a list of nodes. It appears as a subrecord to further describe whichever record precedes it.

Table 8-39 `@NODE`

Format: {@NODE\node list}			
Field	**Type**	**Default**	**Comments**
node list	str	""	Each item in the list specifies a node by its node id.

- Generated by:
  - `probe` statement
  - `probe failures` statement

Example 8-32

```
{@NODE\2|Node53|+5Volts}
```

Record @PCHK: Polarity Check Test

This record describes the results of a test made by an Polarity Check test.

Table 8-40 `@PCHK`

Format: {@PCHK | test status | test designator}			
Field	**Type**	**Default**	**Comments**
test status	int	00	00 passed 01 failed 07 fatal error (test did not complete)
test designator	str	""	

- Generated by: `test` statement

Example 8-33

```
{@PCHK|01|c34}
```

Record @PF: pinsfailed

This record contains the results of a `test` statement. It is followed by a one or more subrecords containing a list of pins.

Table 8-41 `@PF`

Format: {@PF | designator | test status | total pins}			
Field	**Type**	**Default**	**Comments**
designator	str	""	An optional designator that identifies the file containing the source code for the pins test. This comes from the <file id> parameter in the `test` statement.
test status	int	0	0 = testing passed 1 = testing failed
total pins	int	0	Total number of failing pins.

- Generated by: `test` statement
- Subrecords: `@PIN`: Contains a list of pins acquired in `test` mode

Example 8-34

```
{@PF||1|4
   {@PIN\4|10472|12235|21612|11302}
}
```

Record @PIN: list of pins

This record contains a list of pins. It appears as a subrecord to further describe to whichever record precedes it.

Table 8-42 `@PIN`

Format: {@PIN\pin list}			
Field	**Type**	**Default**	**Comments**
pin list	str	""	Each item in the list specifies a pin in BRRCC format.

- Generated by: `pinsfailed` statement

Example 8-35

```
{@PIN\5|11571|20314|12065|20508|11443}
```

Record @PRB: (partial) results of probing (failures)

This record contains the results of the `probe` and `probe failures` statements. It can be followed by subrecords containing a list of failing pins.

Table 8-43 `@PRB`

Format: {@PRB | test status | pin count | test designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 = passed; 1 = failed
pin count	int	0	How many pins failed.
test designator	str	""	The test designator (name of failing device).

- Generated by:
  - `probe` statement
  - `probe failures` statement
- Subrecords: `@DPIN` is generated when logging pins to list failing device pins

Example 8-36

```
{@PRB|1|2|U23
   {@DPIN|\2|Node63||Node22|}
}
```

NOTE One `probe` record is generated for each device tested. Each `probe failures` statement generates subrecords to report all of the failing nodes.

Record @RETEST: indicate a log clear for retest

The presence of this record indicates that a `log clear for retest` occurred. A `log clear for retest` removes data which was logged and later found to be bogus by a `pinsfailed` function being true.

Table 8-44 `@RETEST`

Format: {@RETEST | datetime}			
Field	**Type**	**Default**	**Comments**
datetime	str	""	Date and time of a `log clear for retest` in YYMMDDHHMMSS format (datetime$).

- Generated by: `log clear for retest` statement

Example 8-37

```
{@RETEST|890819184413}
```

Record @RPT: messages logged by report

This record contains the text string produced by executing any of the `report` statements: `report` (ANALOG), `report` (BT-Basic), or `report` (SHORTS) - whenever the report level is `log` or `all`.

Table 8-45 `@RPT`

Format: {@RPT | message}			
Field	**Type**	**Default**	**Comments**
message	str	""	The string given in a `report` statement.

- Generated by: `report` statements

  The location at which this record is nested inside `@BLOCK` records may vary; for example, an `@RPT` record associated with a specific test may appear nested inside the `@BLOCK` record associated with a different test. This causes no problem for the standard software packages that use log data, such as Pushbutton Q-STATS.

Example 8-38

```
{@RPT|U91 failed}
```

Record @TJET: TestJet Test

This record contains the results of either VTEP or TestJet tests.

Table 8-46 `@TJET`

Format: {@TJET | test status | pin count | test designator}			
Field	**Type**	**Default**	**Comments**
test status	int	00	00 = pass 01 = fail 07 = fatal error
pin count	int	0000	number of failing pins
test designator	str	""	

- Generated by: `test` statement
- Subrecords: `@DPIN` is generated when logging pins to list the failing device pins.

Example 8-39

```
{@TJET|01|0008|u34}
```

Record @TS: test shorts

This record describes a shorts test.

Table 8-47 `@TS`

Format: {@TS | test status | shorts count | opens count | phantoms count | designator}			
Field	**Type**	**Default**	**Comments**
test status	int	0	0 passed; 1 failed; 20 learning passed
shorts count	int	0	How many nodes were unexpectedly shorted to some short-source node.
opens count	int	0	How many nodes were unexpectedly open.
phantoms count	int	0	How many phantoms were encountered.
designator	str	""	(optional) Designator that identifies the file containing the source code for the shorts test. This comes from the <file id> parameter in the `test shorts` statement.

- Generated by: `test shorts` statement
- Subrecords:
  - `@TS-O` is generated for each unexpectedly open node pair.
  - `@TS-S` is generated for each unexpectedly shorted short-source node. Contains `@TS-D` subrecords to identify which destination node(s) the source node was shorted to, or `@TS-P` subrecords to identify phantom shorts.

Example 8-40

```
{@TS|1|2|1|1|
   {@TS-S|2|0|Node12
      {@TS-D|Node25|1.678853E+00}
      {@TS-D|Node26|2.543216E+00}
   }
   {@TS-O|Node43|Node14|-1.500000E+00}
   {@TS-S|0|1|Node38
      {@TS-P|-1.243853E+02}
   }
}
```

Record @TS-D: destination nodes shorted to source node

This record describes destination nodes which are shorted to a source node.

Table 8-48 `@TS-D`

Format: {@TS-D\destination list}			
Field	**Type**	**Default**	**Comments**
destination list	str/fp	""/0	A list of items which are paired into two fields for each destination node, as follows: • `node id` the id of this destination node. • `deviation` the difference `M - T`, where `M` is this destination node's measured value and `T` is the threshold.

- Generated by: `test shorts` statement

Example 8-41

```
{@TS-D\4|Node7|1.398537E+02|Node15|4.138792E+01}
```

Record @TS-O: opens found while shorts testing

This record describes open found while shorts testing.

Table 8-49 `@TS-O`

Format: {@TS-O | source node | destination node | deviation}			
Field	**Type**	**Default**	**Comments**
source node	str	""	The open-source node id.
destination node	str	""	The open-destination node id.
deviation	fp	0	The difference `M - T`, where `M` is the measured value and `T` is the threshold.

- Generated by: `test shorts` statement

Example 8-42

```
{@TS-O|Node85|Node14|-1.510000E+00}
```

Record @TS-P: phantoms found while shorts testing

This record describes phantoms found while shorts testing.

Table 8-50 `@TS-P`

Format: {@TS-P | deviation}			
Field	**Type**	**Default**	**Comments**
deviation	fp	0	The difference `M - T`, where `M` is the measured value and `T` is the threshold.

- Generated by: test shorts statement

Example 8-43

```
{@TS-P|-1.243853E+02}
```

Record @TS-S: results of shorts testing from a source node

This record describes a shorts test from a source node.

Table 8-51 `@TS-S`

Format: {@TS-S | shorts count | phantoms count | source node}			
Field	**Type**	**Default**	**Comments**
shorts count	int	0	How many destination nodes were unexpectedly shorted to the source node.
phantoms count	int	0	How many phantoms were encountered from the source node.
source node	str	""	The source node id. If the source is actually a set of `siamese` nodes, then the node id given is representative of the set - probably just the first one in the list.

- Generated by: `test shorts` statement
- Subrecords:
  - `@TS-D` is generated for each destination that is unexpectedly shorted.
  - `@TS-P` is generated for each phantom encountered.

Example 8-44

```
{@TS-S|2||Node43
   {@TS-D|Node14|1.678859E+00}
   {@TS-D|Node32|6.182541E+01}
}
```