Standard Test Data Format (STDF) Specification

Version 4

Table of Contents

Click on any entry.

Introduction to STDF

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™.

A glaring hole in these standards has been the lack of test result data compatibility between test systems of different manufacturers, and sometimes within the product lines of a single manufacturer. In order to help overcome this problem, Teradyne has developed a simple, flexible, portable data format to which existing data files and formats can be easily and economically converted. Called the Standard Test Data Format (STDF™), its specification is contained in the following 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 the test result output of all of its UNIX™ operating system based testers, and offers conversion software for users of its Test System Director for our other semiconductor test systems. Teradyne derives no direct commercial benefit from propagating this standard, but we hope its usefulness, thoroughness, and full documentation will make all of us who work with ATE more productive.

Teradyne's Use of the STDF Specification

The Standard Test Data Format is intended as a comprehensive standard for the entire ATE industry, not as a description of how Teradyne writes or analyzes test result data. A test system can support STDF without using all the STDF record types or filling in all the fields of the record types it does use. Similarly, when the specification says that an STDF record type can be used to create a certain report, it cannot be assumed that Teradyne data analysis software always uses the record type to create its reports. In addition, the statement that a field or record is required or optional applies only to the definition of a valid STDF file; data analysis software may require a field that is declared optional in the specification.

For this reason, the STDF specification is not the final reference on how any piece of Teradyne software implements the specification. To determine how a Teradyne test system fills in the STDF record types, please refer to the documentation for that test system's executive software. To determine what STDF fields are used by a Teradyne data analysis tool, refer to the documentation for the data analysis product.

STDF Design Objectives

As ATE networking continues to emerge into a heterogeneous environment involving various sophisticated computers and operating systems, it becomes necessary to define a common ground that allows testers, database and database management systems, and data analysis software to store and communicate test data in a form that is useful, general, and flexible.

The Standard Test Data Format (STDF) described in this document provides such a form. STDF is flexible enough to meet the needs of the different testers that generate raw test data, the databases that store the data, and the data analysis programs that use the data. The fact that it is a single, coherent standard also facilitates the sharing and communicating of the data among these various components of the complete ATE system.

STDF is not an attempt to specify a database architecture for either testers or the centralized database engines. Instead, it is a set of logical record types. Because data items are described in terms of logical record types, the record types can be used as the underlying data abstraction, whether the data resides in a data buffer, resides on a mass storage device, or is being propagated in a network message. It is independent of network or database architecture. Furthermore, the STDF logical record types may be treated as a convenient data object by any of the software, either networking or database, that may be used on a tester or database engine.

Using a standard but flexible test data format makes it possible for a single data formatting program running on the centralized database engine to accept data from a wide range of testers, whether the testers come from one vendor or from different vendors or are custom-built by the ATE user. In addition, adherence to a standard format permits the exporting of data from the central database and data analysis engine to the user’s in-house network for further analysis in a form that is well documented and thoroughly debugged. Finally, the standard makes it possible to develop portable software for data reporting and analysis on both the testers and the centralized database engine.

STDF Design Objectives

The following list summarizes the major objectives that guided the design of STDF:

- Be capable of storing test data for all semiconductor testers and trimmers.
- Provide a common format for storage and transmission of data.
- Provide a basis for portable data reporting and analysis software.
- Decouple data message format and database format to allow enhancements to either, independently of the other.
- Provide support for optional (missing or invalid) data.
- Provide complete and concise documentation for developers and users.
- Make it easy for customers to write their own reports or reformat data for their own database.

STDF is already a standard within Teradyne:

- All Teradyne semiconductor testers produce raw data in a format that conforms to STDF.
- The Manufacturing Data Pipeline and Insight Series software can process any data written in conformance with STDF.

STDF Record Structure

This section describes the basic STDF record structure. It describes the following general topics, which are applicable to all the record types:

- STDF record header (page 6)
- Record types and subtypes (page 6)
- Data type codes and representation (page 8)
- Optional fields and missing/invalid data (page 11)

STDF Record Header

Each STDF record begins with a record header consisting of the following three fields:

Field	Description
REC_LEN	The number of bytes of data following the record header. REC_LEN does not include the four bytes of the record header.
REC_TYP	An integer identifying a group of related STDF record types.
REC_SUB	An integer identifying a specific STDF record type within each REC_TYP group. On REC_TYP and REC_SUB, see the next section.

Record Types and Subtypes

The header of each STDF record contains a pair of fields called REC_TYP and REC_SUB. Each REC_TYP value identifies a group of related STDF record types. Each REC_SUB value identifies a single STDF record type within a REC_TYP group. The combination of REC_TYP and REC_SUB values uniquely identifies each record type. This design allows groups of related records to be easily identified by data analysis programs, while providing unique identification for each type of record in the file.

All REC_TYP and REC_SUB codes less than 200 are reserved for future use by Teradyne. All codes greater than 200 are available for custom applications use. The codes are all in decimal values. The official list of codes and documentation for their use is maintained by Teradyne's Semiconductor CIM Division (SCD).

The following table lists the meaning of the REC_TYP codes currently defined by Teradyne, as well as the REC_SUB codes defined in the STDF specification.

REC_TYP Code	Meaning and STDF REC_SUB Codes
0	Information about the STDF file 10 File Attributes Record (FAR) 20 Audit Trail Record (ATR)
1	Data collected on a per lot basis 10 Master Information Record (MIR) 20 Master Results Record (MRR) 30 Part Count Record (PCR) 40 Hardware Bin Record (HBR) 50 Software Bin Record (SBR) 60 Pin Map Record (PMR) 62 Pin Group Record (PGR) 63 Pin List Record (PLR) 70 Retest Data Record (RDR) 80 Site Description Record (SDR)
2	Data collected per wafer 10 Wafer Information Record (WIR) 20 Wafer Results Record (WRR) 30 Wafer Configuration Record (WCR)
5	Data collected on a per part basis 10 Part Information Record (PIR) 20 Part Results Record (PRR)
10	Data collected per test in the test program 30 Test Synopsis Record (TSR)
15	Data collected per test execution 10 Parametric Test Record (PTR) 15 Multiple-Result Parametric Record (MPR) 20 Functional Test Record (FTR)
20	Data collected per program segment 10 Begin Program Section Record (BPS) 20 End Program Section Record (EPS)
50	Generic Data 10 Generic Data Record (GDR) 30 Datalog Text Record (DTR)
180	Reserved for use by Image software
181	Reserved for use by IG900 software

Data Type Codes and Representation

The STDF specification uses a set of data type codes that are concise and easily recognizable. For example, R*4 indicates a REAL (float) value stored in four bytes. A byte consists of eight bits of data. For purposes of this document, the low order bit of each byte is designated as bit 0 and the high order bit as bit 7. The following table gives the complete list of STDF data type codes, as well as the equivalent C language type specifier.

Code	Description	C Type Specifier
C*12	Fixed length character string: If a fixed length character string does not fill the entire field, it must be left-justified and padded with spaces.	char[12]
C*n	Variable length character string: first byte = unsigned count of bytes to follow (maximum of 255 bytes)	char[]
C*f	Variable length character string: string length is stored in another field	char[]
U*1	One byte unsigned integer	unsigned char
U*2	Two byte unsigned integer	unsigned short
U*4	Four byte unsigned integer	unsigned long
I*1	One byte signed integer	char
I*2	Two byte signed integer	short
I*4	Four byte signed integer	long
R*4	Four byte floating point number	float
R*8	Eight byte floating point number	long float (double)
B*6	Fixed length bit-encoded data	char[6]
V*n	Variable data type field: The data type is specified by a code in the first byte, and the data follows (maximum of 255 bytes)	
B*n	Variable length bit-encoded field: First byte = unsigned count of bytes to follow (maximum of 255 bytes). First data item in least significant bit of the second byte of the array (first byte is count.)	char[]

Code	Description	C Type Specifier
D*n	Variable length bit-encoded field: First two bytes = unsigned count of bits to follow (maximum of 65,535 bits). First data item in least significant bit of the third byte of the array (first two bytes are count). Unused bits at the high order end of the last byte must be zero.	char[]
N*1	Unsigned integer data stored in a nibble. (Nibble = 4 bits of a byte). First item in low 4 bits, second item in high 4 bits. If an odd number of nibbles is indicated, the high nibble of the byte will be zero. Only whole bytes can be written to the STDF file.	char
*k*x*TYPE*	Array of data of the type specified. The value of '*k*' (the number of elements in the array) is defined in an earlier field in the record. For example, an array of short unsigned integers is defined as ***k*xU*2**.	*TYPE*[]

Note on Time and Date Usage

The date and time field used in this specification is defined as a four byte (32 bit) unsigned integer field measuring the number of seconds since midnight on January 1st, 1970, in the local time zone. This is the UNIX standard base time, adjusted to the local time zone.

Refer to the **Glossary** for definitions of Setup time, Start time, and Finish time as used in STDF.

Note on Data Representation

When data is shared among systems with unlike central processors, the problem arises that there is little or no standardization of data representation (that is, the bit ordering of various data types) among the various processors of the world. For example, the data representations for DEC, Motorola, Intel, and IBM computers are all different, even though at least two of them adhere to the IEEE floating point standard. Moreover, different processors made by the same company sometimes store data in incompatible ways.

To address this problem, the STDF specification uses a field called CPU_TYPE in the File Attributes Record (FAR). This field indicates the type of processor that wrote the data (for example, Sun series or DEC-11 series). The field is used as follows:

- When writing an STDF file, a system uses its own native data representation. The type of the writing processor is stored in the CPU_TYPE field.

- When reading an STDF file, a system must convert the records to its own native data representation as it reads them, if necessary. To do so, it checks the value of the CPU_TYPE field in the FAR, which is the first record in the file. Then, if the writing CPU's data representation is incompatible with its own, it uses a subroutine that reads the next (or selected) record and converts the records to its own data representation as it reads them.

This approach has the following advantages:

- All testers, trimmers, and hosts can read and write local data using their native data representation.

- Testing and local data analysis are not slowed down by performing data conversions on any tester.

- Use of a read subroutine makes data conversion transparent at read time.

This approach works for any combination of host and tester processors, provided that the machines are capable of storing and reading the test data in eight bit bytes.

Optional Fields and Missing/Invalid Data

Certain fields in STDF records are defined as optional. An optional field must be present in the record, but there are ways to indicate that its value is not meaningful, that is, that its data should be considered missing or invalid. There are two such methods:

- Some optional fields have a predefined value that means that the data for the field is missing. For example, if the optional field is a variable-length character string, a length byte of 0 means that the data is missing. If the field is numeric, a value of -1 may be defined as meaning that the data is missing.

- For other optional fields, all possible stored values, including -1, are legal. In this case, the STDF specification for the record defines an Optional Data bit field. Each bit is used to designate whether an optional field in the record contains valid or invalid data. Usually, if the bit for an optional field is set, any data in the field is invalid and should be ignored.

Optional fields at the end of a record may be omitted in order to save space on the storage medium. To be omitted, an optional field must have missing or invalid data, and all the fields following it must be optional fields containing missing or invalid data. It is never legal to omit an optional field from the middle of the record.

The specification of each STDF record has a column labelled **Missing/Invalid Data Flag**. An entry in this column means that the field is optional, and that the value shown is the way to flag the field's data as missing or invalid. If the column does not have an entry, the field is required.

Each data type has a standard way of indicating missing or invalid data, as the following table shows:

Data Type	Missing/Invalid Data Flag
Variable-length string	Set the length byte to 0.
Fixed-length character string	Fill the field with spaces.
Fixed-length binary string	Set a flag bit in an Optional Data byte.
Time and date fields	Use a binary 0.
Signed and unsigned integers and floating point values	Use the indicated reserved value or set a flag bit in an Optional Data byte.

Note on “Required” and “Optional”

The distinction between required and optional fields applies only to the definition of a **minimally valid STDF file**. It is **not** a statement about whether any software (even Teradyne software) requires the field. A field that is marked **optional** in the specification may be **required** by software that reads or analyzes the STDF file, even if Teradyne has written the software.

In most cases, a minimally valid STDF file will not provide sufficient input for a piece of analysis software. You will need to fill in some fields or records that are not marked as required here.

This specification is not intended to define the data requirements for any analysis software. The only authority on whether a piece of software requires a certain STDF field or record is the documentation for that software.

STDF Record Types

This section contains the definitions for the STDF record types. The following information is provided for each record type:

- a statement of function: how the record type is used in the STDF file.
- a table defining the data fields: first the standard STDF header, then the fields specific to this record type. The information includes the field name, the data type (see the previous section for the data type codes), a brief description of the field, and the flag to indicate missing or invalid data (see the previous section for a discussion of optional fields).
- any additional notes on specific fields.
- possible uses for this record type in data analysis reports. Note that this entry states only where the record type can be used. It is not a statement that the reports listed always use this record type, even if Teradyne has written those reports. For definitive information on how any data analysis software uses the STDF file, see the documentation for the data analysis software.
- frequency with which the record type appears in the STDF file: for example, once per lot, once per wafer, one per test, and so forth.
- the location of the record type in the STDF file. See the note on "initial sequence" on the next page.

Note on “Initial Sequence”

For several record types, the “Location” says that the record must appear “after the initial sequence.” The phrase “initial sequence” refers to the records that must appear at the beginning of the STDF file. The requirements for the initial sequence are as follows:

- Every file must contain one **File Attributes Record** (FAR), one **Master Information Record** (MIR), one or more **Part Count Records** (PCR), and one **Master Results Record** (MRR). All other records are optional.
- The **first** record in the STDF file must be the **File Attributes Record** (FAR).
- If one or more **Audit Trail Records** (ATRs) are used, they must appear immediately after the FAR.
- The **Master Information Record** (MIR) must appear in every STDF file. Its location must be after the FAR and the ATRs (if ATRs are used).
- If the **Retest Data Record** (RDR) is used, it must appear immediately after the MIR.
- If one or more **Site Description Records** (SDRs) are used, they must appear immediately after the MIR and RDR (if the RDR is used).

Given these requirements, every STDF record must contain one of these initial sequences:

FAR	–			MIR				
FAR	–	ATRs	–	MIR				
FAR	–			MIR	–	RDR		
FAR	–	ATRs	–	MIR	–	RDR		
FAR	–			MIR	–			SDRs
FAR	–	ATRs	–	MIR	–			SDRs
FAR	–			MIR	–	RDR	–	SDRs
FAR	–	ATRs	–	MIR	–	RDR	–	SDRs

All other STDF record types appear after the initial sequence.

Alphabetical Listing

In this section, the STDF record types appear in order of ascending record type and record subtype codes. For easier reference, the record types are listed on this page in alphabetical order, by the three-letter abbreviations for the record types.

File Attributes Record (FAR)

Function: Contains the information necessary to determine how to decode the STDF data contained in the file.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (0)	
REC_SUB	U*1	Record sub-type (10)	
CPU_TYPE	U*1	CPU type that wrote this file	
STDF_VER	U*1	STDF version number	

Notes on Specific Fields:

CPU_TYPE

Indicates which type of CPU wrote this STDF file. This information is useful for determining the CPU-dependent data representation of the integer and floating point fields in the file's records. The valid values are:

- 0 = DEC PDP-11 and VAX processors. F and D floating point formats will be used. G and H floating point formats will not be used.
- 1 = Sun 1, 2, 3, and 4 computers.
- 2 = Sun 386i computers, and IBM PC, IBM PC-AT, and IBM PC-XT computers.
- 3-127 = Reserved for future use by Teradyne.
- 128-255 = Reserved for use by customers.

A code defined here may also be valid for other CPU types whose data formats are fully compatible with that of the type listed here. Before using one of these codes for a CPU type not listed here, please check with the Teradyne hotline, which can provide additional information on CPU compatibility.

STDF_VER

Identifies the version number of the STDF specification used in generating the data. This allows data analysis programs to handle STDF specification enhancements.

Location: Required as the first record of the file.

Audit Trail Record (ATR)

Function: Used to record any operation that alters the contents of the STDF file. The name of the program and all its parameters should be recorded in the ASCII field provided in this record. Typically, this record will be used to track filter programs that have been applied to the data.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (0)	
REC_SUB	U*1	Record sub-type (20)	
MOD_TIM	U*4	Date and time of STDF file modification	
CMD_LINE	C*n	Command line of program	

Frequency: Optional. One for each filter or other data transformation program applied to the STDF data.

Location: Between the File Attributes Record (FAR) and the Master Information Record (MIR).

The filter program that writes the altered STDF file must write its ATR immediately after the FAR (and hence before any other ATRs that may be in the file). In this way, multiple ATRs will be in reverse chronological order.

Possible Use: Determining whether a particular filter has been applied to the data.

Master Information Record (MIR)

Function: The MIR and the MRR (Master Results Record) contain all the global information that is to be stored for a tested lot of parts. Each data stream must have exactly one MIR, immediately after the FAR (and the ATRs, if they are used). This will allow any data reporting or analysis programs access to this information in the shortest possible amount of time.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (1)	
REC_SUB	U*1	Record sub-type (10)	
SETUP_T	U*4	Date and time of job setup	
START_T	U*4	Date and time first part tested	
STAT_NUM	U*1	Tester station number	
MODE_COD	C*1	Test mode code (e.g. prod, dev)	space
RTST_COD	C*1	Lot retest code	space
PROT_COD	C*1	Data protection code	space
BURN_TIM	U*2	Burn-in time (in minutes)	65,535
CMOD_COD	C*1	Command mode code	space
LOT_ID	C*n	Lot ID (customer specified)	
PART_TYP	C*n	Part Type (or product ID)	
NODE_NAM	C*n	Name of node that generated data	
TSTR_TYP	C*n	Tester type	
JOB_NAM	C*n	Job name (test program name)	
JOB_REV	C*n	Job (test program) revision number	length byte = 0
SBLOT_ID	C*n	Sublot ID	length byte = 0
OPER_NAM	C*n	Operator name or ID (at setup time)	length byte = 0
EXEC_TYP	C*n	Tester executive software type	length byte = 0
EXEC_VER	C*n	Tester exec software version number	length byte = 0
TEST_COD	C*n	Test phase or step code	length byte = 0
TST_TEMP	C*n	Test temperature	length byte = 0
USER_TXT	C*n	Generic user text	length byte = 0
AUX_FILE	C*n	Name of auxiliary data file	length byte = 0
PKG_TYP	C*n	Package type	length byte = 0
FAMLY_ID	C*n	Product family ID	length byte = 0
DATE_COD	C*n	Date code	length byte = 0
FACIL_ID	C*n	Test facility ID	length byte = 0
FLOOR_ID	C*n	Test floor ID	length byte = 0
PROC_ID	C*n	Fabrication process ID	length byte = 0
OPER_FRQ	C*n	Operation frequency or step	length byte = 0

(Continued)

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
SPEC_NAM	C*n	Test specification name	length byte = 0
SPEC_VER	C*n	Test specification version number	length byte = 0
FLOW_ID	C*n	Test flow ID	length byte = 0
SETUP_ID	C*n	Test setup ID	length byte = 0
DSGN_REV	C*n	Device design revision	length byte = 0
ENG_ID	C*n	Engineering lot ID	length byte = 0
ROM_COD	C*n	ROM code ID	length byte = 0
SERL_NUM	C*n	Tester serial number	length byte = 0
SUPR_NAM	C*n	Supervisor name or ID	length byte = 0

Notes on Specific Fields:

MODE_COD

Indicates the station mode under which the parts were tested. Currently defined values for the MODE_COD field are:

A = AEL (Automatic Edge Lock) mode
C = Checker mode
D = Development / Debug test mode
E = Engineering mode (same as Development mode)
M = Maintenance mode
P = Production test mode
Q = Quality Control

All other alphabetic codes are reserved for future use by Teradyne. The characters 0 - 9 are available for customer use.

RTST_COD

Indicates whether the lot of parts has been previously tested under the same test conditions. Suggested values are:

Y = Lot was previously tested.
N = Lot has not been previously tested.
space = Not known if lot has been previously tested.
0 - 9 = Number of times lot has previously been tested.

PROT_COD

User-defined field indicating the protection desired for the test data being stored. Valid values are the ASCII characters 0 - 9 and A - Z. A space in this field indicates a missing value (default protection).

CMOD_COD

Indicates the command mode of the tester during testing of the parts. The user or the tester executive software defines command mode values. Valid values are the ASCII characters 0 - 9 and A - Z. A space indicates a missing value.

TEST_COD	A user-defined field specifying the phase or step in the device testing process.
TST_TEMP	The test temperature is an ASCII string. Therefore, it can be stored as degrees Celsius, Fahrenheit, Kelvin or whatever. It can also be expressed in terms like HOT, ROOM, and COLD if that is preferred.

Frequency: Always required. One per data stream.

Location: Immediately after the File Attributes Record (FAR) and the Audit Trail Records (ATR), if ATRs are used.

Possible Use: Header information for all reports

Master Results Record (MRR)

Function: The Master Results Record (MRR) is a logical extension of the Master Information Record (MIR). The data can be thought of as belonging with the MIR, but it is not available when the tester writes the MIR information. Each data stream must have exactly one MRR as the last record in the data stream.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (1)	
REC_SUB	U*1	Record sub-type (20)	
FINISH_T	U*4	Date and time last part tested	
DISP_COD	C*1	Lot disposition code	space
USR_DESC	C*n	Lot description supplied by user	length byte = 0
EXC_DESC	C*n	Lot description supplied by exec	length byte = 0

Notes on Specific Fields:

DISP_COD	Supplied by the user to indicate the disposition of the lot of parts (or of the tester itself, in the case of checker or AEL data). The meaning of DISP_COD values are user-defined. A valid value is an ASCII alphanumeric character (0 - 9 or A - Z). A space indicates a missing value.

Frequency: Exactly one MRR required per data stream.

Location: Must be the last record in the data stream.

Possible Use:

Final Summary Sheet	Merged Summary Sheet
Datalog	Test Results Synopsis Report
Wafer Map	Trend Plot
Histogram	ADART
Correlation	RTBM
Shmoo Plot	User Data
Repair Report	

Part Count Record (PCR)

Function: Contains the part count totals for one or all test sites. Each data stream must have at least one PCR to show the part count.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (1)	
REC_SUB	U*1	Record sub-type (30)	
HEAD_NUM	U*1	Test head number	See note
SITE_NUM	U*1	Test site number	
PART_CNT	U*4	Number of parts tested	
RTST_CNT	U*4	Number of parts retested	4,294,967,295
ABRT_CNT	U*4	Number of aborts during testing	4,294,967,295
GOOD_CNT	U*4	Number of good (passed) parts tested	4,294,967,295
FUNC_CNT	U*4	Number of functional parts tested	4,294,967,295

Notes on Specific Fields:

HEAD_NUM	If this PCR contains a summary of the part counts for all test sites, this field must be set to 255.
GOOD_CNT, FUNC_CNT	A part is considered good when it is binned into one of the “passing” hardware bins. A part is considered functional when it is good enough to test, whether it passes or not. Parts that are incomplete or have shorts or opens are considered non-functional.

Frequency: There must be at least one PCR in the file: either one summary PCR for all test sites (HEAD_NUM = 255), or one PCR for each head/site combination, or both.

Location: Anywhere in the data stream after the initial sequence (see page 14) and before the MRR. When data is being recorded in real time, this record will usually appear near the end of the data stream.

Possible Use: Final Summary Sheet, Merged Summary Sheet, Site Summary Sheet, Report for Lot Tracking System

Main Menu

Hardware Bin Record (HBR)

Function: Stores a count of the parts "physically" placed in a particular bin after testing. (In wafer testing, "physical" binning is not an actual transfer of the chip, but rather is represented by a drop of ink or an entry in a wafer map file.) This bin count can be for a single test site (when parallel testing) or a total for all test sites. The STDF specification also supports a Software Bin Record (SBR) for logical binning categories. A part is "physically" placed in a hardware bin after testing. A part can be "logically" associated with a software bin during or after testing.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (1)	
REC_SUB	U*1	Record sub-type (40)	
HEAD_NUM	U*1	Test head number	See note
SITE_NUM	U*1	Test site number	
HBIN_NUM	U*2	Hardware bin number	
HBIN_CNT	U*4	Number of parts in bin	
HBIN_PF	C*1	Pass/fail indication	space
HBIN_NAM	C*n	Name of hardware bin	length byte = 0

Notes on Specific Fields:

HEAD_NUM	If this HBR contains a summary of the hardware bin counts for **all** test sites, this field must be set to 255.
HBIN_NUM	Has legal values in the range 0 to 32767.
HBIN_PF	This field indicates whether the hardware bin was a passing or failing bin. Valid values for this field are: P = Passing bin F = Failing bin space = Unknown

Frequency: One per hardware bin for each site. One per hardware bin for bin totals. May be included to name unused bins.

Location: Anywhere in the data stream after the initial sequence (see page 14) and before the MRR. When data is being recorded in real time, this record usually appears near the end of the data stream.

Possible Use:	Final Summary Sheet	Merged Summary Sheet
	Site Summary Sheet	Report for Lot Tracking System

Software Bin Record (SBR)

Function: Stores a count of the parts associated with a particular logical bin after testing. This bin count can be for a single test site (when parallel testing) or a total for all test sites. The STDF specification also supports a Hardware Bin Record (HBR) for actual physical binning. A part is "physically" placed in a hardware bin after testing. A part can be "logically" associated with a software bin during or after testing.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (1)	
REC_SUB	U*1	Record sub-type (50)	
HEAD_NUM	U*1	Test head number	See note
SITE_NUM	U*1	Test site number	
SBIN_NUM	U*2	Software bin number	
SBIN_CNT	U*4	Number of parts in bin	
SBIN_PF	C*1	Pass/fail indication	space
SBIN_NAM	C*n	Name of software bin	length byte = 0

Notes on Specific Fields:

HEAD_NUM	If this SBR contains a summary of the software bin counts for **all** test sites, this field must be set to 255.
SBIN_NUM	Has legal values in the range 0 to 32767.
SBIN_PF	This field indicates whether the software bin was a passing or failing bin. Valid values for this field are: P = Passing bin F = Failing bin space = Unknown

Frequency: One per software bin for each site. One per software bin for bin totals. May be included to name unused bins.

Location: Anywhere in the data stream after the initial sequence (see page 14) and before the MRR. When data is being recorded in real time, this record usually appears near the end of the data stream.

Possible Use:	Final Summary Sheet	Merged Summary Sheet
	Site Summary Sheet	Report for Lot Tracking System

Main Menu

Pin Map Record (PMR)

Function: Provides indexing of tester channel names, and maps them to physical and logical pin names. Each PMR defines the information for a single channel/pin combination. See "Using the Pin Mapping Records" on page 77.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (1)	
REC_SUB	U*1	Record sub-type (60)	
PMR_INDX	U*2	Unique index associated with pin	
CHAN_TYP	U*2	Channel type	0
CHAN_NAM	C*n	Channel name	length byte = 0
PHY_NAM	C*n	Physical name of pin	length byte = 0
LOG_NAM	C*n	Logical name of pin	length byte = 0
HEAD_NUM	U*1	Head number associated with channel	1
SITE_NUM	U*1	Site number associated with channel	1

Notes on Specific Fields:

PMR_INDX	This number is used to associate the channel and pin name information with data in the FTR or MPR. Reporting programs can then look up the PMR index and choose which of the three associated names they will use. The range of legal PMR indexes is 1 - 32,767. The size of the FAIL_PIN and SPIN_MAP arrays in the FTR are directly proportional to the highest PMR index number. Therefore, it is important to start PMR indexes with a low number and use consecutive numbers if possible.
CHAN_TYP	The channel type values are tester-specific. Please refer to the tester documentation for a list of the valid tester channel types and codes.
HEAD_NUM, SITE_NUM	If a test system does not support parallel testing and does not have a standard way of identifying its single test site or head, these fields should be set to 1. If missing, the value of these fields will default to 1.

Frequency: One per channel/pin combination used in the test program.
Reuse of a PMR index number is not permitted.

Location: After the initial sequence (see page 14) and before the first PGR, PLR, FTR, or MPR that uses this record's PMR_INDX value.

Possible Use: Functional Datalog Functional Histogram

Pin Group Record (PGR)

Function: Associates a name with a group of pins. See "Using the Pin Mapping Records" on page 77.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (1)	
REC_SUB	U*1	Record sub-type (62)	
GRP_INDX	U*2	Unique index associated with pin group	
GRP_NAM	C*n	Name of pin group	length byte = 0
INDX_CNT	U*2	Count (*k*) of PMR indexes	
PMR_INDX	*k*xU*2	Array of indexes for pins in the group	INDX_CNT = 0

Notes on Specific Fields:

GRP_INDX	The range of legal group index numbers is 32,768 - 65,535.
INDX_CNT, PMR_INDX	PMR_INDX is an array of PMR indexes whose length is defined by INDX_CNT. The order of the PMR indexes should be from most significant to least significant bit in the pin group (regardless of the order of PMR index numbers).

Frequency: One per pin group defined in the test program.

Location: After all the PMRs whose PMR index values are listed in the PMR_INDX array of this record; and before the first PLR that uses this record's GRP_INDX value.

Possible Use: Functional Datalog

Main Menu

Pin List Record (PLR)

Function: Defines the current display radix and operating mode for a pin or pin group. See “Using the Pin Mapping Records” on page 77.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (1)	
REC_SUB	U*1	Record sub-type (63)	
GRP_CNT	U*2	Count (*k*) of pins or pin groups	
GRP_INDX	*k*xU*2	Array of pin or pin group indexes	
GRP_MODE	*k*xU*2	Operating mode of pin group	0
GRP_RADX	*k*xU*1	Display radix of pin group	0
PGM_CHAR	*k*xC*n	Program state encoding characters	length byte = 0
RTN_CHAR	*k*xC*n	Return state encoding characters	length byte = 0
PGM_CHAL	*k*xC*n	Program state encoding characters	length byte = 0
RTN_CHAL	*k*xC*n	Return state encoding characters	length byte = 0

Notes on Specific Fields:

GRP_CNT	GRP_CNT defines the number of pins or pin groups whose radix and mode are being defined. Therefore, it defines the size of each of the arrays that follow in the record. GRP_CNT must be greater than zero.
GRP_MODE	The following are valid values for the pin group mode: 00 = Unknown 10 = Normal 20 = SCIO (Same Cycle I/O) 21 = SCIO Midband 22 = SCIO Valid 23 = SCIO Window Sustain 30 = Dual drive (two drive bits per cycle) 31 = Dual drive Midband 32 = Dual drive Valid 33 = Dual drive Window Sustain Unused pin group modes in the range of 1 through 32,767 are reserved for future use. Pin group modes 32,768 through 65,535 are available for customer use.

GRP_RADX	The following are valid values for the pin group display radix: 0 = Use display program default 2 = Display in Binary 8 = Display in Octal 10 = Display in Decimal 16 = Display in Hexadecimal 20 = Display as symbolic
PGM_CHAR, PGM_CHAL	These ASCII characters are used to display the programmed state in the FTR or MPR. Use of these character arrays makes it possible to store tester-dependent display representations in a tester-independent format. If a single character is used to represent each programmed state, then only the PGM_CHAR array need be used. If two characters represent each state, then the first (left) character is stored in PGM_CHAL and the second (right) character is stored in PGM_CHAR.
RTN_CHAR, RTN_CHAL	These ASCII characters are used to display the returned state in the FTR or MPR. Use of these character arrays makes it possible to store tester-dependent display representations in a tester-independent format. If a single character is used to represent each returned state, then only the RTN_CHAR array need be used. If two characters represent each state, then the first (left) character is stored in RTN_CHAL and the second (right) character is stored in RTN_CHAR.

Note on Missing/Invalid Data Flags:

For each field, the missing/invalid data flag applies to each member of the array, not to the array as a whole. Empty arrays (or empty members of arrays) can be omitted if they occur at the end of the record. Otherwise, each array must have the number of members indicated by GRP_CNT. You can then use the field's missing/invalid data flag to indicate which array members have no data. For example, if GRP_CNT = 3, and if PGM_CHAL contains no data (but RTN_CHAL, which appears after PGM_CHAL, does), then PGM_CHAL should be an array of three missing/invalid data flags: 0, 0, 0.

Frequency: One or more whenever the usage of a pin or pin group changes in the test program.

Location: After all the PMRs and PGRs whose PMR index values and pin group index values are listed in the GRP_INDX array of this record; and before the first FTR that references pins or pin groups whose modes are defined in this record.

Possible Use: Functional Datalog

Retest Data Record (RDR)

Function: Signals that the data in this STDF file is for retested parts. The data in this record, combined with information in the MIR, tells data filtering programs what data to replace when processing retest data.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (1)	
REC_SUB	U*1	Record sub-type (70)	
NUM_BINS	U*2	Number (*k*) of bins being retested	
RTST_BIN	*k*xU*2	Array of retest bin numbers	NUM_BINS = 0

Notes on Specific Fields:

NUM_BINS, RTST_BIN	NUM_BINS indicates the number of hardware bins being retested and therefore the size of the RTST_BIN array that follows. If NUM_BINS is zero, then all bins in the lot are being retested and RTST_BIN is omitted. The LOT_ID, SUBLOT_ID, and TEST_COD of the current STDF file should match those of the STDF file that is being retested, so the data can be properly merged at a later time.

Frequency: Optional. One per data stream.

Location: If this record is used, it must appear immediately after the Master Information Record (MIR).

Possible Use: Tells data filtering programs how to handle retest data.

Site Description Record (SDR)

Function: Contains the configuration information for one or more test sites, connected to one test head, that compose a site group.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (1)	
REC_SUB	U*1	Record sub-type (80)	
HEAD_NUM	U*1	Test head number	
SITE_GRP	U*1	Site group number	
SITE_CNT	U*1	Number (*k*) of test sites in site group	
SITE_NUM	*k*xU*1	Array of test site numbers	
HAND_TYP	C*n	Handler or prober type	length byte = 0
HAND_ID	C*n	Handler or prober ID	length byte = 0
CARD_TYP	C*n	Probe card type	length byte = 0
CARD_ID	C*n	Probe card ID	length byte = 0
LOAD_TYP	C*n	Load board type	length byte = 0
LOAD_ID	C*n	Load board ID	length byte = 0
DIB_TYP	C*n	DIB board type	length byte = 0
DIB_ID	C*n	DIB board ID	length byte = 0
CABL_TYP	C*n	Interface cable type	length byte = 0
CABL_ID	C*n	Interface cable ID	length byte = 0
CONT_TYP	C*n	Handler contactor type	length byte = 0
CONT_ID	C*n	Handler contactor ID	length byte = 0
LASR_TYP	C*n	Laser type	length byte = 0
LASR_ID	C*n	Laser ID	length byte = 0
EXTR_TYP	C*n	Extra equipment type field	length byte = 0
EXTR_ID	C*n	Extra equipment ID	length byte = 0

Notes on Specific Fields:

SITE_GRP	Specifies a site group number (called a station number on some testers) for the group of sites whose configuration is defined by this record. Note that this is different from the station number specified in the MIR, which refers to a software station only. The value in this field must be unique within the STDF file.
SITE_CNT, SITE_NUM	SITE_CNT tells how many sites are in the site group that the current SDR configuration applies to. SITE_NUM is an array of those site numbers.

_TYP fields	These are the type or model number of the interface or peripheral equipment being used for testing: HAND_TYP, CARD_TYP, LOAD_TYP, DIB_TYP, CABL_TYP, CONT_TYP, LASR_TYP, EXTR_TYP
_ID fields	These are the IDs or serial numbers of the interface or peripheral equipment being used for testing: HAND_ID, CARD_ID, LOAD_ID, DIB_ID, CABL_ID, CONT_ID, LASR_ID, EXTR_ID

Frequency: One for each site or group of sites that is differently configured.

Location: Immediately after the MIR and RDR (if an RDR is used).

Possible Use: Correlation of yield to interface or peripheral equipment

Wafer Information Record (WIR)

Function: Acts mainly as a marker to indicate where testing of a particular wafer begins for each wafer tested by the job plan. The WIR and the Wafer Results Record (WRR) bracket all the stored information pertaining to one tested wafer. This record is used only when testing at wafer probe. A WIR/WRR pair will have the same HEAD_NUM and SITE_GRP values.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (2)	
REC_SUB	U*1	Record sub-type (10)	
HEAD_NUM	U*1	Test head number	
SITE_GRP	U*1	Site group number	255
START_T	U*4	Date and time first part tested	
WAFER_ID	C*n	Wafer ID	length byte = 0

Notes on Specific Fields:

SITE_GRP	Refers to the site group in the SDR. This is a means of relating the wafer information to the configuration of the equipment used to test it. If this information is not known, or the tester does not support the concept of site groups, this field should be set to 255.
WAFER_ID	Is optional, but is strongly recommended in order to make the resultant data files as useful as possible.

Frequency: One per wafer tested.

Location: Anywhere in the data stream after the initial sequence (see page 14) and before the MRR.
Sent before testing each wafer.

Possible Use: Wafer Summary Sheet Datalog
Wafer Map

Wafer Results Record (WRR)

Function: Contains the result information relating to each wafer tested by the job plan. The WRR and the Wafer Information Record (WIR) bracket all the stored information pertaining to one tested wafer. This record is used only when testing at wafer probe time. A WIR/WRR pair will have the same HEAD_NUM and SITE_GRP values.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (2)	
REC_SUB	U*1	Record sub-type (20)	
HEAD_NUM	U*1	Test head number	
SITE_GRP	U*1	Site group number	255
FINISH_T	U*4	Date and time last part tested	
PART_CNT	U*4	Number of parts tested	
RTST_CNT	U*4	Number of parts retested	4,294,967,295
ABRT_CNT	U*4	Number of aborts during testing	4,294,967,295
GOOD_CNT	U*4	Number of good (passed) parts tested	4,294,967,295
FUNC_CNT	U*4	Number of functional parts tested	4,294,967,295
WAFER_ID	C*n	Wafer ID	length byte = 0
FABWF_ID	C*n	Fab wafer ID	length byte = 0
FRAME_ID	C*n	Wafer frame ID	length byte = 0
MASK_ID	C*n	Wafer mask ID	length byte = 0
USR_DESC	C*n	Wafer description supplied by user	length byte = 0
EXC_DESC	C*n	Wafer description supplied by exec	length byte = 0

Notes on Specific Fields:

SITE_GRP	Refers to the site group in the SDR. This is a means of relating the wafer information to the configuration of the equipment used to test it. If this information is not known, or the tester does not support the concept of site groups, this field should be set to 255.
WAFER_ID	Is optional, but is strongly recommended in order to make the resultant data files as useful as possible. A Wafer ID in the WRR supersedes any Wafer ID found in the WIR.
FABWF_ID	Is the ID of the wafer when it was in the fabrication process. This facilitates tracking of wafers and correlation of yield with fabrication variations.
FRAME_ID	Facilitates tracking of wafers once the wafer has been through the saw step and the wafer ID is no longer readable on the wafer itself. This is an important piece of information for implementing an inkless binning scheme.

Frequency: One per wafer tested.

Location: Anywhere in the data stream after the corresponding WIR.
Sent after testing each wafer.

Possible Use:

Wafer Summary Sheet	Datalog
Wafer Map	

Wafer Configuration Record (WCR)

Function: Contains the configuration information for the wafers tested by the job plan. The WCR provides the dimensions and orientation information for all wafers and dice in the lot. This record is used only when testing at wafer probe time.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (2)	
REC_SUB	U*1	Record sub-type (30)	
WAFR_SIZ	R*4	Diameter of wafer in WF_UNITS	0
DIE_HT	R*4	Height of die in WF_UNITS	0
DIE_WID	R*4	Width of die in WF_UNITS	0
WF_UNITS	U*1	Units for wafer and die dimensions	0
WF_FLAT	C*1	Orientation of wafer flat	space
CENTER_X	I*2	X coordinate of center die on wafer	-32768
CENTER_Y	I*2	Y coordinate of center die on wafer	-32768
POS_X	C*1	Positive X direction of wafer	space
POS_Y	C*1	Positive Y direction of wafer	space

Notes on Specific Fields:

WF_UNITS	Has these valid values:	0 = Unknown units 1 = Units are in inches 2 = Units are in centimeters 3 = Units are in millimeters 4 = Units are in mils
WF_FLAT	Has these valid values:	U = Up D = Down L = Left R = Right space = Unknown
CENTER_X, CENTER_Y	Use the value -32768 to indicate that the field is invalid.	

POS_X	Has these valid values:	L	=	Left
		R	=	Right
		space	=	Unknown
POS_Y	Has these valid values:	U	=	Up
		D	=	Down
		space	=	Unknown

Frequency: One per STDF file (used only if wafer testing).

Location: Anywhere in the data stream after the initial sequence (see page 14), and before the MRR.

Possbile Use: Wafer Map

Part Information Record (PIR)

Function: Acts as a marker to indicate where testing of a particular part begins for each part tested by the test program. The PIR and the Part Results Record (PRR) bracket all the stored information pertaining to one tested part.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (5)	
REC_SUB	U*1	Record sub-type (10)	
HEAD_NUM	U*1	Test head number	
SITE_NUM	U*1	Test site number	

Notes on Specific Fields:

HEAD_NUM, SITE_NUM	If a test system does not support parallel testing, and does not have a standard way to identify its single test site or head, then these fields should be set to 1. When parallel testing, these fields are used to associate individual datalogged results (FTRs and PTRs) with a PIR/PRR pair. An FTR or PTR belongs to the PIR/PRR pair having the same values for HEAD_NUM and SITE_NUM.

Frequency: One per part tested.

Location: Anywhere in the data stream after the initial sequence (see page 14), and before the corresponding PRR.
Sent before testing each part.

Possible Use: Datalog

Part Results Record (PRR)

Function: Contains the result information relating to each part tested by the test program. The PRR and the Part Information Record (PIR) bracket all the stored information pertaining to one tested part.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (5)	
REC_SUB	U*1	Record sub-type (20)	
HEAD_NUM	U*1	Test head number	
SITE_NUM	U*1	Test site number	
PART_FLG	B*1	Part information flag	
NUM_TEST	U*2	Number of tests executed	
HARD_BIN	U*2	Hardware bin number	
SOFT_BIN	U*2	Software bin number	65535
X_COORD	I*2	(Wafer) X coordinate	-32768
Y_COORD	I*2	(Wafer) Y coordinate	-32768
TEST_T	U*4	Elapsed test time in milliseconds	0
PART_ID	C*n	Part identification	length byte = 0
PART_TXT	C*n	Part description text	length byte = 0
PART_FIX	B*n	Part repair information	length byte = 0

Notes on Specific Fields:

HEAD_NUM, SITE_NUM	If a test system does not support parallel testing, and does not have a standard way to identify its single test site or head, then these fields should be set to 1. When parallel testing, these fields are used to associate individual datalogged results (FTRs and PTRs) with a PIR/PRR pair. An FTR or PTR belongs to the PIR/PRR pair having the same values for HEAD_NUM and SITE_NUM.
X_COORD, Y_COORD	Have legal values in the range -32767 to 32767. A missing value is indicated by the value -32768.
X_COORD, Y_COORD, PART_ID	Are all optional, but you should provide either the PART_ID or the X_COORD and Y_COORD in order to make the resultant data useful for analysis.

PART_FLG	Contains the following fields: bit 0: 0 = This is a new part. Its data device does not supersede that of any previous device. 1 = The PIR, PTR, MPR, FTR, and PRR records that make up the current sequence (identified as having the same HEAD_NUM and SITE_NUM) supersede any previous sequence of records with the same PART_ID. (A repeated part sequence usually indicates a mistested part.) bit 1: 0 = This is a new part. Its data device does not supersede that of any previous device. 1 = The PIR, PTR, MPR, FTR, and PRR records that make up the current sequence (identified as having the same HEAD_NUM and SITE_NUM) supersede any previous sequence of records with the same X_COORD and Y_COORD. (A repeated part sequence usually indicates a mistested part.) **Note:** Either Bit 0 or Bit 1 can be set, but **not both**. (It is also valid to have neither set.) bit 2: 0 = Part testing completed normally 1 = Abnormal end of testing bit 3: 0 = Part passed 1 = Part failed bit 4: 0 = Pass/fail flag (bit 3) is valid 1 = Device completed testing with no pass/fail indication (i.e., bit 3 is invalid) bits 5 - 7: Reserved for future use — must be 0
HARD_BIN	Has legal values in the range 0 to 32767.
SOFT_BIN	Has legal values in the range 0 to 32767. A missing value is indicated by the value 65535.
PART_FIX	This is an application-specific field for storing device repair information. It may be used for bit-encoded, integer, floating point, or character information. Regardless of the information stored, the first byte must contain the number of bytes to follow. This field can be decoded only by an application-specific analysis program. See “Storing Repair Information” on page 75.

Frequency: One per part tested.

Location: Anywhere in the data stream after the corresponding PIR and before the MRR. Sent after completion of testing each part.

Possible Use:

Datalog	Wafer map
RTBM	Shmoo Plot
Repair Data	

Main Menu

Test Synopsis Record (TSR)

Function: Contains the test execution and failure counts for one parametric or functional test in the test program. Also contains static information, such as test name. The TSR is related to the Functional Test Record (FTR), the Parametric Test Record (PTR), and the Multiple Parametric Test Record (MPR) by test number, head number, and site number.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (10)	
REC_SUB	U*1	Record sub-type (30)	
HEAD_NUM	U*1	Test head number	See note
SITE_NUM	U*1	Test site number	
TEST_TYP	C*1	Test type	space
TEST_NUM	U*4	Test number	
EXEC_CNT	U*4	Number of test executions	4,294,967,295
FAIL_CNT	U*4	Number of test failures	4,294,967,295
ALRM_CNT	U*4	Number of alarmed tests	4,294,967,295
TEST_NAM	C*n	Test name	length byte = 0
SEQ_NAME	C*n	Sequencer (program segment/flow) name	length byte = 0
TEST_LBL	C*n	Test label or text	length byte = 0
OPT_FLAG	B*1	Optional data flag	See note
TEST_TIM	R*4	Average test execution time in seconds	OPT_FLAG bit 2 = 1
TEST_MIN	R*4	Lowest test result value	OPT_FLAG bit 0 = 1
TEST_MAX	R*4	Highest test result value	OPT_FLAG bit 1 = 1
TST_SUMS	R*4	Sum of test result values	OPT_FLAG bit 4 = 1
TST_SQRS	R*4	Sum of squares of test result values	OPT_FLAG bit 5 = 1

Notes on Specific Fields:

HEAD_NUM	If this TSR contains a summary of the test counts for all test sites, this field must be set to 255.
TEST_TYP	Indicates what type of test this summary data is for. Valid values are: P = Parametric test F = Functional test M = Multiple-result parametric test space = Unknown

EXEC_CNT, FAIL_CNT, ALRM_CNT	Are optional, but are strongly recommended because they are needed to compute values for complete final summary sheets.
OPT_FLAG	Contains the following fields: bit 0 set = TEST_MIN value is invalid bit 1 set = TEST_MAX value is invalid bit 2 set = TEST_TIM value is invalid bit 3 is reserved for future use and must be 1 bit 4 set = TST_SUMS value is invalid bit 5 set = TST_SQRS value is invalid bits 6 - 7 are reserved for future use and must be 1 OPT_FLAG is optional if it is the last field in the record.
TST_SUMS, TST_SQRS	Are useful in calculating the mean and standard deviation for a single lot or when combining test data from multiple STDF files.

Frequency: One for each test executed in the test program.
May optionally be used to identify unexecuted tests.

Location: Anywhere in the data stream after the initial sequence (see page 14) and before the MRR.
When test data is being generated in real-time, these records will appear after the last PRR.

Possible Use:

Final Summary Sheet	Datalog
Merged Summary Sheet	Histogram
Wafer Map	Functional Histogram

Parametric Test Record (PTR)

Function: Contains the results of a single execution of a parametric test in the test program. The first occurrence of this record also establishes the default values for all semi-static information about the test, such as limits, units, and scaling. The PTR is related to the Test Synopsis Record (TSR) by test number, head number, and site number.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (15)	
REC_SUB	U*1	Record sub-type (10)	
TEST_NUM	U*4	Test number	
HEAD_NUM	U*1	Test head number	
SITE_NUM	U*1	Test site number	
TEST_FLG	B*1	Test flags (fail, alarm, etc.)	
PARM_FLG	B*1	Parametric test flags (drift, etc.)	
RESULT	R*4	Test result	TEST_FLG bit 1 = 1
TEST_TXT	C*n	Test description text or label	length byte = 0
ALARM_ID	C*n	Name of alarm	length byte = 0
OPT_FLAG	B*1	Optional data flag	See note
RES_SCAL	I*1	Test results scaling exponent	OPT_FLAG bit 0 = 1
LLM_SCAL	I*1	Low limit scaling exponent	OPT_FLAG bit 4 or 6 = 1
HLM_SCAL	I*1	High limit scaling exponent	OPT_FLAG bit 5 or 7 = 1
LO_LIMIT	R*4	Low test limit value	OPT_FLAG bit 4 or 6 = 1
HI_LIMIT	R*4	High test limit value	OPT_FLAG bit 5 or 7 = 1
UNITS	C*n	Test units	length byte = 0
C_RESFMT	C*n	ANSI C result format string	length byte = 0
C_LLMFMT	C*n	ANSI C low limit format string	length byte = 0
C_HLMFMT	C*n	ANSI C high limit format string	length byte = 0
LO_SPEC	R*4	Low specification limit value	OPT_FLAG bit 2 = 1
HI_SPEC	R*4	High specification limit value	OPT_FLAG bit 3 = 1

Notes on Specific Fields:

Default Data	All data following the OPT_FLAG field has a special function in the STDF file. The first PTR for each test will have these fields filled in. These values will be the default for each subsequent PTR with the same test number: if a subsequent PTR has a value for one of these fields, it will be used instead of the default, for that one record only; if the field is blank, the default will be used. This method replaces use of the PDR in STDF V3. If the PTR is not associated with a test execution (that is, it contains only default information), bit 4 of the TEST_FLG field must be set, and the PARM_FLG field must be zero. Unless the default is being overridden, the default data fields should be omitted in order to save space in the file. Note that RES_SCAL, LLM_SCAL, HLM_SCAL, UNITS, C_RESFMT, C_LLMFMT, and C_HLMFMT are interdependent. If you are overriding the default value of one, make sure that you also make appropriate changes to the others in order to keep them consistent. For character strings, you can override the default with a null value by setting the string length to 1 and the string itself to a single binary 0.
HEAD_NUM, SITE_NUM	If a test system does not support parallel testing, and does not have a standard way of identifying its single test site or head, these fields should be set to 1. When parallel testing, these fields are used to associate individual datalogged results with a PIR/PRR pair. A PTR belongs to the PIR/PRR pair having the same values for HEAD_NUM and SITE_NUM.

TEST_FLG	Contains the following fields: bit 0: 0 = No alarm 1 = Alarm detected during testing bit 1: 0 = The value in the RESULT field is valid (see note on RESULT) 1 = The value in the RESULT field is not valid. This setting indicates that the test was executed, but no datalogged value was taken. You should read bits 6 and 7 of TEST_FLG to determine if the test passed or failed. bit 2: 0 = Test result is reliable 1 = Test result is unreliable bit 3: 0 = No timeout 1 = Timeout occurred bit 4: 0 = Test was executed 1 = Test not executed bit 5: 0 = No abort 1 = Test aborted bit 6: 0 = Pass/fail flag (bit 7) is valid 1 = Test completed with no pass/fail indication bit 7: 0 = Test passed 1 = Test failed
PARM_FLG	Is the parametric flag field, and contains the following bits: bit 0: 0 = No scale error 1 = Scale error bit 1: 0 = No drift error 1 = Drift error (unstable measurement) bit 2: 0 = No oscillation 1 = Oscillation detected bit 3: 0 = Measured value not high 1 = Measured value higher than high test limit bit 4: 0 = Measured value not low 1 = Measured value lower than low test limit bit 5: 0 = Test failed or test passed standard limits 1 = Test passed alternate limits bit 6: 0 = If result = low limit, then result is “fail.” 1 = If result = low limit, then result is “pass.” bit 7: 0 = If result = high limit, then result is “fail.” 1 = If result = high limit, then result is “pass.”

RESULT	The RESULT value is considered useful only if **all** the following bits from TEST_FLG and PARM_FLG are 0: TEST_FLG bit 0 = 0 no alarm bit 1 = 0 value in result field is valid bit 2 = 0 test result is reliable bit 3 = 0 no timeout bit 4 = 0 test was executed bit 5 = 0 no abort PARM_FLG bit 0 = 0 no scale error bit 1 = 0 no drift error bit 2 = 0 no oscillation If any one of these bits is 1, then the PTR result should **not** be used.
ALARM_ID	If the alarm flag (bit 0 of TEST_FLG) is set, this field can contain the name or ID of the alarms that were triggered. Alarm names are tester-dependent.
OPT_FLAG	Is the Optional data flag and contains the following bits: bit 0 set = RES_SCAL value is invalid. The default set by the first PTR with this test number will be used. bit 1 reserved for future used and must be 1. bit 2 set = No low specification limit. bit 3 set = No high specification limit. bit 4 set = LO_LIMIT and LLM_SCAL are invalid. The default values set for these fields in the first PTR with this test number will be used. bit 5 set = HI_LIMIT and HLM_SCAL are invalid. The default values set for these fields in the first PTR with this test number will be used. bit 6 set = No Low Limit for this test (LO_LIMIT and LLM_SCAL are invalid). bit 7 set = No High Limit for this test (HI_LIMIT and HLM_SCAL are invalid). The OPT_FLAG field may be omitted if it is the last field in the record.
C_RESFMT, C_LLMFMT, C_HLMFMT	ANSI C format strings for use in formatting the test result and low and high limits (both test and spec). For example, “%7.2f”.The format string is also known as an output specification string, as used with the printf statement. See any ANSI C reference man, or the man page on printf.
LO_SPEC, HI_SPEC	The specification limits are set in the first PTR and should never change. They use the same scaling and format strings as the corresponding test limits.

Frequency: One per parametric test execution.

Location: Under normal circumstances, the PTR can appear anywhere in the data stream after the corresponding Part Information Record (PIR) and before the corresponding Part Result Record (PRR).

In addition, to facilitate conversion from STDF V3, if the first PTR for a test contains default information only (no test results), it may appear anywhere after the initial sequence (see page 14), and before the first corresponding PTR, but need not appear between a PIR and PRR.

Possible Use: Datalog
Histogram
Wafer Map

Storing and Displaying Parametric Test Data:

The values stored as RESULT, LO_LIMIT, HI_LIMIT, LO_SPEC, and HI_SPEC are all normalized to the base unit stored as UNITS. The UNITS text string indicates base (whole) units only, with no scaling factor: for example, UNITS may be “AMPS” or “VOLTS” but never “uAMPS” or “mVOLTS”. Therefore, the UNITS value provides enough information to represent the stored result or limit. In addition, because of this normalization, arithmetic can be performed directly on any values for which the UNITS fields agree.

In displaying a result or limit, however, it is sometimes desirable to use a scale other than the base units: for example, “uAMPS” rather than “AMPS”. It is also desirable to indicate the precision to which the value was measured. Scaling and precision are indicated by using additional fields.

Scaling uses the RES_SCAL, LLM_SCAL and HLM_SCAL fields. The _SCAL value is an integer that indicates the power of ten of the scaling factor:

scaled result =	`RESULT * (10 ** RES_SCAL)`
scaled low limit =	`LO_LIMIT * (10 ** LLM_SCAL)`
scaled high limit =	`HI_LIMIT * (10 ** HLM_SCAL)`

Storing and Displaying Parametric Test Data (continued):

The _SCAL value also serves as a code that indicates the prefix to be added to the UNITS value in order to obtain the correctly scaled units. The meaning of the codes is given in the following table, which defines the recognized values for the RES_SCAL, HLM_SCAL, and LLM_SCAL fields:

_SCAL **value**	UNITS **Prefix**	**Meaning**	**Magnitude**
15	f	femto	10**-15
12	p	pico	10**-12
9	n	nano	10**-9
6	u	micro	10**-6
3	m	milli	10**-3
2	%	percent	10**-2
0			10**0
-3	K	Kilo	10**3
-6	M	Mega	10**6
-9	G	Giga	10**9
-12	T	Tera	10**12

For example, if UNITS is AMPS and RES_SCAL is 6, the display units are uAMPS.

In order to **display** a result or limit, the C_RESFMT, C_LLMFMT, and C_HLMFMT fields may be used as appropriate. These provide an ANSI C compatible format string for displaying the result or limit. This string should provide all the information necessary to output the string with the correct precision using a format compatible with the data being collected.

For example, to store the result value “123.45 uAMPS,” make the following assignments:

RESULT	123.4517*10**(-6)	(trailing “17” is rounding error)
RES_SCAL	6	(“micro” has the code 6)
C_RESFMT	%7.2f	(minimum field width of 7, precision of 2)
UNITS	AMPS	(the base units)

Again, notice that the RESULT and UNITS alone correctly represent the value, and that RES_SCAL and C_RESFMT are important only when it comes time to display the result. In this example, to display the result one would multiply RESULT by 10**6, display two digits to the right of the decimal point in a total field width of 7, look up the RES_SCAL value of “6” to determine the prefix “u”, and display the UNITS:

```
123.45 uAMPS
```

Multiple-Result Parametric Record (MPR)

Function: Contains the results of a single execution of a parametric test in the test program where that test returns multiple values. The first occurrence of this record also establishes the default values for all semi-static information about the test, such as limits, units, and scaling. The MPR is related to the Test Synopsis Record (TSR) by test number, head number, and site number.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (15)	
REC_SUB	U*1	Record sub-type (15)	
TEST_NUM	U*4	Test number	
HEAD_NUM	U*1	Test head number	
SITE_NUM	U*1	Test site number	
TEST_FLG	B*1	Test flags (fail, alarm, etc.)	
PARM_FLG	B*1	Parametric test flags (drift, etc.)	
RTN_ICNT	U*2	Count (*j*) of PMR indexes	See note
RSLT_CNT	U*2	Count (*k*) of returned results	See note
RTN_STAT	*j*xN*1	Array of returned states	RTN_ICNT = 0
RTN_RSLT	*k*xR*4	Array of returned results	RSLT_CNT = 0
TEST_TXT	C*n	Descriptive text or label	length byte = 0
ALARM_ID	C*n	Name of alarm	length byte = 0
OPT_FLAG	B*1	Optional data flag	See note
RES_SCAL	I*1	Test result scaling exponent	OPT_FLAG bit 0 = 1
LLM_SCAL	I*1	Test low limit scaling exponent	OPT_FLAG bit 4 or 6 = 1
HLM_SCAL	I*1	Test high limit scaling exponent	OPT_FLAG bit 5 or 7 = 1
LO_LIMIT	R*4	Test low limit value	OPT_FLAG bit 4 or 6 = 1
HI_LIMIT	R*4	Test high limit value	OPT_FLAG bit 5 or 7 = 1
START_IN	R*4	Starting input value (condition)	OPT_FLAG bit 1 = 1
INCR_IN	R*4	Increment of input condition	OPT_FLAG bit 1 = 1
RTN_INDX	*j*xU*2	Array of PMR indexes	RTN_ICNT = 0
UNITS	C*n	Units of returned results	length byte = 0
UNITS_IN	C*n	Input condition units	length byte = 0
C_RESFMT	C*n	ANSI C result format string	length byte = 0
C_LLMFMT	C*n	ANSI C low limit format string	length byte = 0
C_HLMFMT	C*n	ANSI C high limit format string	length byte = 0
LO_SPEC	R*4	Low specification limit value	OPT_FLAG bit 2 = 1
HI_SPEC	R*4	High specification limit value	OPT_FLAG bit 3 = 1

Notes on Specific Fields:

Default Data	All data beginning with the OPT_FLAG field has a special function in the STDF file. The first MPR for each test will have these fields filled in. These values will be the default for each subsequent MPR with the same test number: if a subsequent MPR has a value for one of these fields, it will be used instead of the default, for that one record only; if the field is blank, the default will be used. If the MPR is not associated with a test execution (that is, it contains only default information), bit 4 of the TEST_FLG field must be set, and the PARM_FLG field must be zero. Unless the default is being overridden, the default data fields should be omitted in order to save space in the file. Note that RES_SCAL, LLM_SCAL, HLM_SCAL, UNITS, C_RESFMT, C_LLMFMT, and C_HLMFMT are interdependent. If you are overriding the default value of one, make sure that you also make appropriate changes to the others in order to keep them consistent. For character strings, you can override the default with a null value by setting the string length to 1 and the string itself to a single binary 0.
TEST_NUM	The test number does not implicitly increment for successive values in the result array.
HEAD_NUM, SITE_NUM	If a test system does not support parallel testing, and does not have a standard way of identifying its single test site or head, these fields should be set to 1. When parallel testing, these fields are used to associate individual datalogged results with a PIR/PRR pair. An MPR belongs to the PIR/PRR pair having the same values for HEAD_NUM and SITE_NUM.
TEST_FLG	Contains the following fields: bit 0: 0 = No alarm 1 = Alarm detected during testing bit 1: Reserved for future use. Must be zero. bit 2: 0 = Test results are reliable 1 = Test results are unreliable bit 3: 0 = No timeout 1 = Timeout occurred bit 4: 0 = Test was executed 1 = Test not executed bit 5: 0 = No abort 1 = Test aborted bit 6: 0 = Pass/fail flag (bit 7) is valid 1 = Test completed with no pass/fail indication bit 7: 0 = Test passed 1 = Test failed

PARM_FLG	Is the parametric flag field, and contains the following bits: bit 0: 0 = No scale error 1 = Scale error bit 1: 0 = No drift error 1 = Drift error (unstable measurement) bit 2: 0 = No oscillation 1 = Oscillation detected bit 3: 0 = Measured value not high 1 = Measured value higher than high test limit bit 4: 0 = Measured value not low 1 = Measured value lower than low test limit bit 5: 0 = Test failed or test passed standard limits 1 = Test passed alternate limits bit 6: 0 = If result = low limit, then result is “fail.” 1 = If result = low limit, then result is “pass.” bit 7: 0 = If result = high limit, then result is “fail.” 1 = If result = high limit, then result is “pass.”
RTN_ICNT, RTN_INDX, RTN_STAT	The number of element in the RTN_INDX and RTN_STAT arrays is determined by the value of RTN_ICNT. The RTN_STAT field is stored 4 bits per value. The first value is stored in the low order 4 bits of the byte. If the number of indexes is odd, the high order 4 bits of the last byte in RTN_STAT will be padded with zero. The indexes referred to in the RTN_INDX are the PMR indexes defined in the Pin Map Record (PMR). The return state codes are the same as those defined for the RTN_STAT field in the FTR. RTN_ICNT may be omitted if it is zero and it is the last field in the record.
RSLT_CNT, RTN_RSLT	RSLT_CNT defines the number of parametric test results in the RTN_RSLT. If this is a multiple pin measurement, and if PMR indexes will be specified, then the value of RSLT_CNT should be the same as RTN_ICNT. RTN_RSLT is an array of the parametric test result values. RSLT_CNT may be omitted if it is zero and it is the last field in the record.
ALARM_ID	If the alarm flag (bit 0 of TEST_FLG) is set, this field can contain the name or ID of the alarms that were triggered. Alarm names are tester-dependent.

OPT_FLAG	Is the Optional Data Flag and contains the following bits: bit 0 set = RES_SCAL value is invalid. The default set by the first MPR with this test number will be used. bit 1 set = START_IN and INCR_IN are invalid. bit 2 set = No low specification limit. bit 3 set = No high specification limit. bit 4 set = LO_LIMIT and LLM_SCAL are invalid. The default values set for these fields in the first MPR with this test number will be used. bit 5 set = HI_LIMIT and HLM_SCAL are invalid. The default values set for these fields in the first MPR with this test number will be used. bit 6 set = No Low Limit for this test (LO_LIMIT and LLM_SCAL are invalid). bit 7 set = No High Limit for this test (HI_LIMIT and HLM_SCAL are invalid). The OPT_FLAG field may be omitted if it is the last field in the record.
START_IN, INCR_IN, UNITS_IN	For logging shmoo data, these fields specify the input conditions. START_IN is the beginning input value and INCR_IN is the increment, in UNITS_IN units. The input is applied and the output measured RSLT_CNT number of times. Values for INCR_IN can be positive or negative.
LO_LIMIT, HI_LIMIT, UNITS	Regardless of how many test measurements are made, all must use the same limits, units, scaling, and significant digits.
C_RESFMT, C_LLMFMT, C_HLMFMT	ANSI C format strings for use in formatting the test result and low and high limits (both test and spec). For example, “%7.2f”.The format string is also known as an output specification string, as used with the printf statement. See any ANSI C reference man, or the man page on printf.
LO_SPEC, HI_SPEC	The specification limits are set in the first MPR and should never change. They use the same scaling and format strings as the corresponding test limits.

Frequency: One per multiple-result parametric test execution.

Location: Anywhere in the data stream after the corresponding Part Information Record (PIR) and before the corresponding Part Result Record (PRR).

Possible Use: Datalog Shmoo Plot

Functional Test Record (FTR)

Function: Contains the results of the single execution of a functional test in the test program. The first occurrence of this record also establishes the default values for all semi-static information about the test. The FTR is related to the Test Synopsis Record (TSR) by test number, head number, and site number.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (15)	
REC_SUB	U*1	Record sub-type (20)	
TEST_NUM	U*4	Test number	
HEAD_NUM	U*1	Test head number	
SITE_NUM	U*1	Test site number	
TEST_FLG	B*1	Test flags (fail, alarm, etc.)	
OPT_FLAG	B*1	Optional data flag	See note
CYCL_CNT	U*4	Cycle count of vector	OPT_FLAG bit 0 = 1
REL_VADR	U*4	Relative vector address	OPT_FLAG bit 1 = 1
REPT_CNT	U*4	Repeat count of vector	OPT_FLAG bit 2 = 1
NUM_FAIL	U*4	Number of pins with 1 or more failures	OPT_FLAG bit 3 = 1
XFAIL_AD	I*4	X logical device failure address	OPT_FLAG bit 4 = 1
YFAIL_AD	I*4	Y logical device failure address	OPT_FLAG bit 4 = 1
VECT_OFF	I*2	Offset from vector of interest	OPT_FLAG bit 5 = 1
RTN_ICNT	U*2	Count (*j*) of return data PMR indexes	See note
PGM_ICNT	U*2	Count (*k*) of programmed state indexes	See note
RTN_INDX	*j*xU*2	Array of return data PMR indexes	RTN_ICNT = 0
RTN_STAT	*j*xN*1	Array of returned states	RTN_ICNT = 0
PGM_INDX	*k*xU*2	Array of programmed state indexes	PGM_ICNT = 0
PGM_STAT	*k*xN*1	Array of programmed states	PGM_ICNT = 0
FAIL_PIN	D*n	Failing pin bitfield	length bytes = 0
VECT_NAM	C*n	Vector module pattern name	length byte = 0
TIME_SET	C*n	Time set name	length byte = 0
OP_CODE	C*n	Vector Op Code	length byte = 0
TEST_TXT	C*n	Descriptive text or label	length byte = 0
ALARM_ID	C*n	Name of alarm	length byte = 0
PROG_TXT	C*n	Additional programmed information	length byte = 0
RSLT_TXT	C*n	Additional result information	length byte = 0
PATG_NUM	U*1	Pattern generator number	255
SPIN_MAP	D*n	Bit map of enabled comparators	length byte = 0

Notes on Specific Fields:

Default Data	All data starting with the PATG_NUM field has a special function in the STDF file. The first FTR for each test will have these fields filled in. These values will be the default for each subsequent FTR with the same test number. If a subsequent FTR has a value for one of these fields, it will be used instead of the default, for that one record only. If the field is blank, the default will be used. This method replaces use of the FDR in STDF V3. Unless the default is being overridden, the default data fields should be omitted in order to save space in the file.
HEAD_NUM, SITE_NUM	If a test system does not support parallel testing, and does not have a standard way of identifying its single test site or head, these fields should be set to 1. When parallel testing, these fields are used to associate individual datalogged results with a PIR/PRR pair. An FTR belongs to the PIR/PRR pair having the same values for HEAD_NUM and SITE_NUM.
TEST_FLG	Contains the following fields: bit 0: 0 = No alarm 1 = Alarm detected during testing bit 1: Reserved for future use — must be 0 bit 2: 0 = Test result is reliable 1 = Test result is unreliable bit 3: 0 = No timeout 1 = Timeout occurred bit 4: 0 = Test was executed 1 = Test not executed bit 5: 0 = No abort 1 = Test aborted bit 6: 0 = Pass/fail flag (bit 7) is valid 1 = Test completed with no pass/fail indication bit 7: 0 = Test passed 1 = Test failed
OPT_FLAG	Contains the following fields: bit 0 set = CYCL_CNT data is invalid bit 1 set = REL_VADR data is invalid bit 2 set = REPT_CNT data is invalid bit 3 set = NUM_FAIL data is invalid bit 4 set = XFAIL_AD and YFAIL_AD data are invalid bit 5 set = VECT_OFF data is invalid (offset defaults to 0) bits 6 - 7 are reserved for future use and must be 1 This field is only optional if it is the last field in the record.

XFAIL_AD, YFAIL_AD	The logical device address produced by the memory pattern generator, before going through conversion to a physical memory address. This logical address can be different from the physical address presented to the DUT pins.
VECT_OFF	This is the integer offset of this vector (in sequence of execution) from the vector of interest (usually the failing vector). For example, if this FTR contains data for the vector before the vector of interest, this field is set to -1. If this FTR contains data for the third vector after the vector of interest, this field is set to 3. If this FTR is the vector of interest, VECT_OFF is set to 0. It is therefore possible to record an entire sequence of vectors around a failing vector for use with an offline debugger or analysis program.
RTN_ICNT, PGM_ICNT	These fields may be omitted if all data following them is missing or invalid.
RTN_ICNT, RTN_INDX, RTN_STAT	The size of the RTN_INDX and RTN_STAT arrays is determined by the value of RTN_ICNT. The RTN_STAT field is stored 4 bits per value. The first value is stored in the low order 4 bits of the byte. If the number of indexes is odd, the high order 4 bits of the last byte in RTN_STAT will be padded with zero. The indexes referred to in the RTN_INDX are those defined in the PMR.
RTN_STAT	The table of valid returned state values (expressed as hexadecimal digits) is: 0 = 0 or low 1 = 1 or high 2 = midband 3 = glitch 4 = undetermined 5 = failed low 6 = failed high 7 = failed midband 8 = failed with a glitch 9 = open A = short The characters generated to represent these values are tester-dependent, and are specified in the PLR.
PGM_ICNT, PGM_INDX, PGM_STAT	The size of the PGM_INDX and PGM_STAT arrays is determined by the value of PGM_ICNT. The indexes referred to in the PGM_INDX are those defined in the PMR.

PGM_STAT	The table of valid program state values (expressed in hexadecimal) is listed below. Note that there are three defined program modes: Normal, Dual Drive (two drive bits per cycle), and SCIO (same cycle I/O). The characters generated to represent these values are tester-dependent, and are specified in the PLR.

Normal Mode Program States		Typical State Representation
0 =	Drive Low	0
1 =	Drive High	1
2 =	Expect Low	L
3 =	Expect High	H
4 =	Expect Midband	M
5 =	Expect Valid (not midband)	V
6 =	Don't drive, or compare.	X
7 =	Keep window open from prior cycle. (used to "stretch" a comparison across cycles)	W

Dual Drive Mode Program States		Typical State Representations	
0 =	Low at D2, Low at D1 times	00	0
1 =	Low at D2, High at D1 times	10	1
2 =	Hi at D2, Low at D1 times	01	2
3 =	Hi at D2, High at D1 times	11	3
4 =	Compare Low	L	
5 =	Compare High	H	
6 =	Compare Midband	M	
7 =	Don't Compare	X	

SCIO Mode Program States		Typical State Representations	
0 =	Drive Low, Compare Low.	0L	l
1 =	Drive Low, Compare High	0H	h
2 =	Drive Low, Compare Midband	0M	m
3 =	Drive Low, Don't Compare	0X	x
4 =	Drive High, Compare Low.	1L	L
5 =	Drive High, Compare High	1H	H
6 =	Drive High, Compare Midband	1M	M
7 =	Drive High, Don't Compare	1X	X

FAIL_PIN	Encoded with PMR index 0 in bit 0 of the field, PMR index 1 in the 1st position, and so on. Bits representing PMR indexes of failing pins are set to 1.

ALARM_ID	If the alarm flag (bit 0 of TEST_FLG) is set, this field can optionally contain the name or ID of the alarm or alarms that were triggered. The names of these alarms are tester-dependent.
SPIN_MAP	This field contains an array of bits corresponding to the PMR index numbers of the enabled comparators. The 0th bit corresponds to PMR index 0, the 1st bit corresponds to PMR index 1, and so on. Each comparator that is enabled will have its corresponding PMR index bit set to 1.

Frequency: One or more for each execution of a functional test.

Location: Anywhere in the data stream after the corresponding Part Information Record (PIR) and before the corresponding Part Result Record (PRR).

Possible Use: Datalog, Functional Histogram, Functional Failure Analyzer

Begin Program Section Record (BPS)

Function: Marks the beginning of a new program section (or sequencer) in the job plan.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (20)	
REC_SUB	U*1	Record sub-type (10)	
SEQ_NAME	C*n	Program section (or sequencer) name	length byte = 0

Frequency: Optional on each entry into the program segment.

Location: Anywhere after the PIR and before the PRR.

Possible Use: When performing analyses on a particular program segment's test.

End Program Section Record (EPS)

Function: Marks the end of the current program section (or sequencer) in the job plan.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (20)	
REC_SUB	U*1	Record sub-type (20)	

Frequency: Optional on each exit from the program segment.

Location: Following the corresponding BPS and before the PRR in the data stream.

Possible Use:

When performing analyses on a particular program segment's test.

Note that pairs of BPS and EPS records can be nested: for example, when one sequencer calls another. In this case, the sequence of records could look like this:

BPS	SEQ_NAME = sequence-1
BPS	SEQ_NAME = sequence-2
EPS	(*end of* sequence-2)
EPS	(*end of* sequence-1)

Because an EPS record does not contain the name of the sequencer, it should be assumed that each EPS record matches the last unmatched BPS record.

Generic Data Record (GDR)

Function: Contains information that does not conform to any other record type defined by the STDF specification. Such records are intended to be written under the control of job plans executing on the tester. This data may be used for any purpose that the user desires.

Data Fields:

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (50)	
REC_SUB	U*1	Record sub-type (10)	
FLD_CNT	U*2	Count of data fields in record	
GEN_DATA	V*n	Data type code and data for one field *(Repeat* GEN_DATA *for each data field)*	

Notes on Specific Fields:

GEN_DATA Is repeated FLD_CNT number of times. Each GEN_DATA field consists of a data type code followed by the actual data. The data type code is the first unsigned byte of the field. Valid data types are:

0	=	B*0	Special pad field, of length 0 (See note below)
1	=	U*1	One byte unsigned integer
2	=	U*2	Two byte unsigned integer
3	=	U*4	Four byte unsigned integer
4	=	I*1	One byte signed integer
5	=	I*2	Two byte signed integer
6	=	I*4	Four byte signed integer
7	=	R*4	Four byte floating point number
8	=	R*8	Eight byte floating point number
10	=	C*n	Variable length ASCII character string (first byte is string length in bytes)
11	=	B*n	Variable length binary data string (first byte is string length in bytes)
12	=	D*n	Bit encoded data (first two bytes of string are length in bits)
13	=	N*1	Unsigned nibble

Pad Field (Data Type 0):

Data type 0, the special pad field, is used to force alignment of following data types in the record. In particular, it must be used to ensure even byte alignment of U*2, U*4, I*2, I*4, R*4, and R*8 data types.

The GDR is guaranteed to begin on an even byte boundary. The GDR header contains four bytes. The first GEN_DATA field therefore begins on an even byte boundary. It is the responsibility of the designer of a GDR record to provide the pad bytes needed to ensure data boundary alignment for the CPU on which it will run.

Example: The following table describes a sample GDR that contains three data fields of different data types. The assumption is that numeric data of more than one byte must begin on an even boundary. Pad bytes will be used to meet this requirement.

Data	Code	Alignment Requirement
"AB"	10	A variable-length character string can begin on any byte. This field will contain one data byte, one length byte, and two data bytes, for a total length of 4 bytes. Because this field begins on an even byte, the next field also begins on an even byte.
255	1	A one-byte numeric value can begin on any byte. This field contains two bytes, so the next field also begins on an even byte.
510	5	A two-byte numeric value must begin on an even byte. This GEN_DATA field would begin on an even byte; and, because the first byte is the data code, the actual numeric value would begin on an odd byte. This field must therefore be preceded by a pad byte.

The byte representation for this GDR is as follows. The byte ordering shown here is for sample purposes only. The actual data representation differs between CPUs. The byte values are shown in hexadecimal. The decimal equivalents are given in the description of the bytes.

Even Byte	Odd Byte	Description (with Decimal Values)
0c	00	Number of bytes following the header (12)
32	0a	Record type (50); record subtype (10)
04	00	Number of data fields (4)
0a	02	Character string: code (10) and length (2)
41	42	Character string: data bytes (“A” and “B”)
01	ff	1-byte integer: code (1) and data (255 = 0xff)
00	05	Pad byte (0); code (5) for next field
fe	01	2-byte signed integer (510 = 0x01fe)

Frequency: A test data file may contain any number of GDRs.

Location: Anywhere in the data stream after the initial sequence (see page 14).

Possible Use: User-written reports

Datalog Text Record (DTR)

Function: Contains text information that is to be included in the datalog printout. DTRs may be written under the control of a job plan: for example, to highlight unexpected test results. They may also be generated by the tester executive software: for example, to indicate that the datalog sampling rate has changed. DTRs are placed as comments in the datalog listing.

Data Fields:

Field Name	Data Type	Field Description		Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header		
REC_TYP	U*1	Record type	(50)	
REC_SUB	U*1	Record sub-type	(30)	
TEXT_DAT	C*n	ASCII text string		

Frequency: A test data file may contain any number of DTRs.

Location: Anywhere in the data stream after the initial sequence (see page 14).

Possible Use: Datalog

STDF Filenames

An STDF file name must have the following format:

`filename.STD[string]`

where

filename Is any string consisting of 1 to 39 of the ASCII characters A - Z, a - z, and 0 - 9, plus the underscore (_). The first character must be alphabetic. Users should be aware that, while some operating systems distinguish between uppercase and lowercase characters, most do not.

.STD*[string]* Is a string beginning with the characters `.STD`, and continuing with characters that are legal for *filename*. The string cannot be longer than 39 characters. Under systems that support file extensions, this is the file extension. Under system that do not, it is considered to be a fixed literal string. For systems that distinguish between uppercase and lowercase characters, this string should be in lowercase (`.std`).

Note these points:

- In previous versions of the specification, the dollar sign ($) was a legal filename character. It is no longer supported, because its use is incompatible with certain operating systems.
- The STDF filename can contain only a single period. Software that processes STDF files may check for an extension, which is defined as the string after the first period. Many operating systems permit only one period per filename.

(continued)

STDF Filenames

- Use only the characters defined as legal for *filename*. This restricted set is intended to be compatible with as many operating systems and software packages as possible. Using other characters may have unforeseen consequences: for example, some data analysis software may not accept a filename containing a character that you used.

- It is strongly recommended that you use only .STD, without any additional string for the extension. If you must add additional characters, add as few as possible. Software that processes STDF files may add characters to the .STD extension to indicate the state of processing. To avoid exceeding system-specific limits, it is best if the original filename extension is as short as possible, i.e., .STD.

- Some software that processes STDF files retains only the part of the filename to the left of the period (the filename part, not the .STD extension part). It is therefore recommended that the filename to the left of the period be unique, to ensure that the names remain unique after other software has processed the file.

The goals for choosing your STDF file names should be as follows:

- to provide unique file names throughout a system

- to indicate the data contained in the file

- to indicate when the test data was generated

- to provide some level of customer control

- to work on a variety of computer systems

STDF File Ordering

Test data collected by testers is usually written directly to files in STDF format. Each STDF file contains the test data for one lot of parts. To make the data management software efficient and reliable, it is important that all the raw test data for a single insertion of a single lot be stored in one STDF data file.

The STDF test data file must contain one FAR, one MIR, at least one PCR, and one MRR. All other records are optional. The file may therefore contain any combination of datalog, summary, and site summary for that lot of parts.

Data records in the STDF file may be arranged in a variety of ways. The following factors can affect the record ordering:

- whether wafers are being tested
- whether parallel testing is in effect
- whether test description records are being used
- whether datalogging is in effect

The following pages show different ways in which the STDF format can be used to store test data.

1. **For a lot of packaged devices:**

Global information for the file .... FAR
Global information for entire lot .... MIR

Testing each part:

Information on first tested part .... PIR
Results of first test on first part .... PTR/MPR/FTR
.
.
Results of final test on first part .... PTR/MPR/FTR
Final results on first tested part .... PRR

Information on second tested part .... PIR
Repeat test suite for second part .... PTR/MPR/FTR
.
.
Final results on second tested part .... PRR

Repeat sequence for each tested part .... PIR
.
.
.
PRR

Final results for entire lot:

Summary (count of test executions, count of failed parts, etc.) for each test in job plan (one TSR per test in plan) .... TSR
.
.
TSR

Count of parts placed in each hardware bin .... HBR
(one HBR per hardware bin)
.
.
HBR

Count of parts assigned to each logical bin .... SBR
(one SBR per software bin)
.
.
SBR

Part count totals for lot .... PCR
Global summary of results for entire lot .... MRR

2. For a lot of devices for which only summary information is required:

Global information for the file .... FAR
Global information for entire lot .... MIR

Final results for entire lot:

Summary (count of test executions, count of failed parts, etc.) for each test in job plan (one TSR per test in plan) .... TSR
.
.
TSR

Count of parts placed in each hardware bin (one HBR per hardware bin) .... HBR
.
.
HBR

Count of parts assigned to each logical bin (one SBR per software bin) .... SBR
.
.
SBR

Part count totals for lot .... PCR
Global summary of results for entire lot .... MRR

Main Menu

3. For a lot of devices at wafer probe:

Global information for the file .... FAR
Global information for entire lot .... MIR
Dimensions and orientation of wafer .... WCR

Testing each wafer:

Information for first wafer .... WIR
Information for first die .... PIR
Perform test suite on first die..................................... PTR/MPR/FTR
.
.
Final results of test suite on first die......................... PRR

Repeat for each die of first wafer .... PIR
PTR/MPR/FTR
.
.
PRR

Test results summary for dice of first wafer .... WRR

Repeat sequence for each remaining wafer .... WIR
.
.
WRR

Final results for entire lot:

Summary (count of test executions, count of failed parts, etc.) for each test in job plan (one TSR per test in plan) .... TSR
.
.
TSR

Count of dice placed in each hardware bin (one HBR per hardware bin) .... HBR
.
.
HBR

Count of dice placed in each logical bin (one SBR per software bin) .... SBR
.
.
SBR

Part count totals for lot .... PCR
Global summary of results for entire lot .... MRR

4. For a lot of devices, storing only information necessary to generate a wafer map:

Global information for the file .... FAR
Global information for entire lot .... MIR
Dimensions and orientation of wafer .... WCR

Testing each wafer:

Information for wafer .... WIR
Information for first die .... PIR
Final results of test suite on first die .... PRR

Repeat for each remaining die .... PIR
PRR
.
.

Test results summary for all dice of wafer .... WRR

Repeat sequence for remaining wafers .... WIR
.
.
WRR

Final results for entire lot:

Summary (count of test executions, count of failed parts, etc.) for each test in job plan (one TSR per test in plan) .... TSR
.
.
TSR

Count of dice placed in each hardware bin (one HBR per hardware bin) .... HBR
.
.
HBR

Count of dice placed in each logical bin (one SBR per software bin) .... SBR
.
.
SBR

Part count totals for lot .... PCR
Global summary of results for entire lot .... MRR

5. For a lot of devices tested at wafer probe on a parallel tester (two test heads, two sites per head):

Global information for the file ....FAR
Global information for entire lot ....MIR
Dimensions and orientation of wafer ....WCR

Testing first two wafers (one per test head):

Information for first wafer (head 1) ....WIR, head 1
Information for second wafer (head 2) ....WIR, head 2
Beginning of first die ....PIR, head 1 site 1
Beginning of second die...PIR, head 1 site 2
Beginning of third die...PIR, head 2 site 1
Beginning of fourth die..PIR, head 2 site 2
First test on first die...PTR/MPR/FTR, head 1 site 1
First test on second die ....PTR/MPR/FTR, head 1 site 2
First test on third die ....PTR/MPR/FTR, head 2 site 1
First test on fourth die ....PTR/MPR/FTR, head 2 site 2

Second test on first die...PTR/MPR/FTR, head 1 site 1
Second test on second die ....PTR/MPR/FTR, head 1 site 2
Second test on third die ....PTR/MPR/FTR, head 2 site 1
Second test on fourth die ....PTR/MPR/FTR, head 2 site 2

Repeat test suite on all four dice ....PTR/MPR/FTR ...
.
.
Test suite finishes first on second die;......................PRR, head 1 site 2
this PRR has results of all tests on this die.

Finish test suite on remaining three dice.................PTR/MPR/FTR, head 1 site 1
PTR/MPR/FTR, head 2 site 1
PTR/MPR/FTR, head 2 site 2
.
.
Results of all tests on first die ....PRR, head 1 site 1
Results of all tests on third die...................................PRR, head 2 site 1
Results of all tests on fourth die ....PRR, head 2 site 2

Repeat sequence for next set of four dice ....PIR ...
.
.
PRR ...

(continued)

5. For a lot of devices tested at wafer probe on a parallel tester (continued):

Information for final four dice .... PIR, head 1 site 1
PIR, head 1 site 2
PIR, head 2 site 1
PIR, head 2 site 2

Perform each test on each die...................................... PTR/MPR/FTR ...
.
.

Test suite finishes on first and second dice; all tests now complete on head 1 dice .... PRR, head 1 site 1
PRR, head 1 site 2

Test results for all dice of head 1 wafer .... WRR, head 1

Perform remaining tests on head 2 dice.................... PTR/MPR/FTR, head 2 site 1
PTR/MPR/FTR, head 2 site 2
.
.

Test suite finishes on third and fourth dice; all tests now complete on head 2 dice .... PRR, head 2 site 1
PRR, head 2 site 2

Test results for all dice of head 2 wafer .... WRR, head 2

Testing remaining wafers:

Repeat sequence for each set of wafers .... WIR, head 1
WIR, head 2
.
.

Test results for all dice of next wafers .... WRR, head 1
WRR, head 2

(continued)

5. **For a lot of devices tested at wafer probe on a parallel tester (continued):**

Final results for entire lot:

Summary (count of test executions, count of failed parts, etc.) for each test in job plan (one TSR per test in plan) ....TSR
.
.

Count of dice placed in each hardware bin ....HBR
(one HBR per hardware bin) .
.

Count of dice assigned to each logical bin ....SBR
(one SBR per software bin) .
.

Part count for head 1, site 1 ....PCR
Part count for head 1, site 2 ....PCR
Part count for head 2, site 1 ....PCR
Part count for head 2, site 2 ....PCR
Part count for entire lot (HEAD_NUM = 255).............PCR

Global summary of results for entire lot...........MRR

Storing Repair Information

Data for repair of memory, PC boards, and other parts can be stored and passed between the test and repair processes using the STDF format. The repair information for each part tested is stored in the PART_FIX field in the Part Results Record (PRR).

It is possible to keep repair data in the same STDF file as all the other test information or to separate it out in order to minimize the number of bytes passed from one process to the next. The following examples are intended to provide additional help in understanding how the STDF records are used in storing repair information. Additional STDF records may be used in the file for more information as desired.

Storing Repair Information

The following is the ordering of the minimum records required for an STDF file containing memory repair data from wafer probe:

File Attributes Record...FAR
Master Information Record..MIR
    Wafer information for 1st wafer...............................WIR
        Part results for 1st device on wafer..........................PRR
        Part results for 2nd device on wafer ....PRR
        .
        .
        Part results for last device on wafer ....PRR
    Wafer results for 1st wafer...WRR
    .
    .
    Wafer information for last wafer...............................WIR
        Part results for 1st device on wafer..........................PRR
        Part results for 2nd device on wafer ....PRR
        .
        .
        Part results for last device on wafer ....PRR
    Wafer results for last wafer.......................................WRR
Part Count Record ....PCR
Master Results Record...MRR

The following is the ordering of the minimum records required for an STDF file containing PC board repair data from a board tester:

File Attributes Record...FAR
Master Information Record..MIR
    Part results for 1st PC board ....PRR
    Part results for 2nd PC board....................................PRR
    .
    .
    Part results for last PC board....................................PRR
Part Count Record ....PCR
Master Results Record...MRR

Using the Pin Mapping Records

When testing devices, either packaged or as part of a wafer, there is a mapping between device pins and tester channels. This mapping is defined in the **Pin Map Record** (PMR). Each channel will have a type and a name. Each pin will have a physical and a logical name. The PMR defines one unique association between a channel and a pin and assigns that mapping a number, known as the PMR Index. These indexes are in the range 1 -32,767.

Pins are sometimes defined in groups, such as address pins, or data pins. The **Pin Group Record** (PGR) allows a group of pins to be named and given a group index number. The PGR lists the PMR Indexes for the pins in a pin group and assigns a name and a Group Index number to that group. Group index numbers are in the range 32-768 - 65,535.

For any pin group, there is a display radix and an operating mode. For groups of pins that are multiplexed (i.e., that serve multiple functions at different times), there may be more than one set of radixes and operating modes. Depending on the tester type and potentially the device type, there may also be different data representations for those modes. The **Pin List Record** (PLR) defines a mapping between one or more pins (by Pin Index) and/or pin groups (by Group Index) and their corresponding display radixes and operating modes. It also defines the programmed-state and returned-state character representations for those pins.

Both the Functional Test Record (FTR) and the Multiple-Result Parametric Record (MPR) use the Pin Indexes defined in the PMR to associate state information with their corresponding pins. Both programmed states and returned states can be decoded and displayed using information from the PMRs, PGRs, and PLRs associated with the Pin Indexes listed in the FTRs and MPRs.

Using the Pin Mapping Records

Software that decodes and displays functional test data will use the Pin Index mapping in the FTRs and MPRs to determine which pins had what values. The software can use the PMR to determine the physical and logical name of the pin, the channel name and type associated with that pin, and which test head and site that channel is connected to. Using data from the PGR, the software will be able to determine whether the pin is part of a pin group and, if so, what the group name is and what other pins are part of that group. Data from the PLR will then be able to tell the software how data associated with that pin should be displayed to make it understandable to engineers and programmers dealing with that type of tester and device.

Differences Between STDF V3 and V4

Since its introduction in 1985, Teradyne's Standard Test Data Format (STDF) has gained wide-spread acceptance, so much so that it has become a de facto standard in the ATE industry. In using STDF over the years, customers have found that it meets many of their data needs. Inevitably, however, their intensive use of STDF revealed places where they needed additional fields or different structures.

Teradyne has listened to these customers, and has collected nearly one hundred requests and comments from twenty-two customers and six ATE vendors, as well as its own engineers who have been using STDF. The result is the first new version of STDF in years: STDF Version 4.

This section summarizes the differences between STDF V3 and V4. It first lists the record types of the two versions. It then lists the changes to the data types used in defining the STDF records. Finally, it lists the changes to each STDF record type, and indicates which record types have remained unchanged.

For details on any of these differences, see the rest of the STDF specification.

Record Types

The following table shows all of the V3 and V4 record types. Codes in regular font are in both V3 and V4. Codes in **bold** are new in V4. Codes in ***italic*** are in V3, but have been dropped from V4.

REC_TYP	**Meaning and STDF** REC_SUB **Code**	
0	Information about the STDF file	
	10	File Attributes Record (FAR)
	20	**Audit Trail Record (ATR) – New**
1	Data collected on a per lot basis	
	10	Master Information Record (MIR)
	20	Master Results Record (MRR)
	30	**Part Count Record (PCR) – New**
	40	Hardware Bin Record (HBR)
	50	Software Bin Record (SBR)
	60	Pin Map Record (PMR)
	62	**Pin Group Record (PGR) – New**
	63	**Pin List Record (PLR) – New**
	70	**Retest Data Record (RDR) – New**
	80	**Site Description Record (SDR) – New**
2	Data collected per wafer	
	10	Wafer Information Record (WIR)
	20	Wafer Results Record (WRR)
	30	Wafer Configuration Record (WCR)
5	Data collected on a per part basis	
	10	Part Information Record (PIR)
	20	Part Results Record (PRR)
10	Data collected per test in the test program	
	10	***Parametric Test Description Record (PDR) – Dropped***
	20	***Functional Test Description Record (FDR) – Dropped***
	30	Test Synopsis Record (TSR)
15	Data collected per test execution	
	10	Parametric Test Record (PTR)
	15	**Multiple-Result Parametric Record (MPR) – New**
	20	Functional Test Record (FTR)
20	Data collected per program segment	
	10	Begin Program Section Record (BPS)
	20	End Program Section Record (EPS)

REC_TYP	Meaning and STDF REC_SUB Code
25	Data collected per test site — ***All Dropped***
	10 Site-Specific Hardware Bin Record (SHB)
	20 Site-Specific Software Bin Record (SSB)
	30 Site-Specific Test Synopsis Record (STS)
	40 Site-Specific Part Count Record (SCR)
50	Generic Data
	10 Generic Data Record (GDR)
	30 Datalog Text Record (DTR)

Data Types

The following change has been made for V4:

B*n — First data item is now in least significant bit of the second byte of the array (first byte is count.)

The following data types have been **added** to V4.

D*n — Variable length bit-encoded field:
First 2 bytes = unsigned count of bits to follow (max. of 65,535 bits).
First data item in least significant bit of the third byte of the array.
Unused bits at the high order end of the last byte must be zero.

N*1 — Unsigned integer data stored in a nibble. (Nibble = 4 bits of a byte).
First item in low 4 bits, second item in high 4 bits.
For an odd number of nibbles, the high nibble of the byte will be zero. Only whole bytes can be written to the STDF file.

*k*x*TYPE* — Array of data of the type specified.
The value of '*k*' (the number of elements in the array) is defined in an earlier field.
For example, an array of short unsigned integers is defined as ***k*xU*2**.

Filename Characters

The dollar sign ($) is no longer a valid character in an STDF filename. The **only** valid characters are the alphanumerics and the underscore (_).

Required Records

Under V3, the only required records in an STDF file were the MIR and MRR.

Under V4, there are **four** required records:

FAR	The first record in the must be the FAR. There is exactly one FAR per file.
MIR	There must be exactly one MIR per file. The MIR must follow the FAR and any ATRs (if they are used).
PCR	There must be at least one PCR per file: either one summary PCR (HEAD_NUM = 255), or one PCR per head/site combination, or both. The PCRs must come after the MIR and before the MRR.
MRR	There must be exactly one MRR per file. It must be the final record in the file.

Changes to Specific STDF Record Types

ATR: Audit Trail Record — New in V4

Records any operation (such as a filter program) that alters the contents of the STDF file. If these records are used, they must immediately follow the FAR.

Data Fields (after header):

MOD_TIM	Date and time of STDF file modification
CMD_LINE	Command line of program that altered the file

MIR: Master Information Record

First Record in File:

Under V3, the first record in the STDF file could be an FAR or an MIR.
Under V4, the first record must be an FAR.

MIR Fields Added for V4:

BURN_TIM	Burn-in time (in minutes)
EXEC_VER	Tester exec software version number
TST_TEMP	Test temperature
USER_TXT	Generic user text
AUX_FILE	Name of auxiliary data file
PKG_TYP	Package type
FAMLY_ID	Product family ID
DATE_COD	Date code
FACIL_ID	Test facility ID
FLOOR_ID	Test floor ID
OPER_FRQ	Operation frequency or step
SPEC_NAM	Test specification name
SPEC_VER	Test specification version number
FLOW_ID	Test flow ID
SETUP_ID	Test setup ID
DSGN_REV	Device design revision
ENG_ID	Engineering lot ID
ROM_COD	ROM code ID
SERL_NUM	Tester serial number

V3 Fields Dropped From V4:

CPU_TYPE	Now only in FAR
STDF_VER	Now only in FAR
HAND_ID	Moved to SDR (Site Description Record – new in V4)
PRB_CARD	Moved to SDR (as CARD_ID)

Other MIR Changes:

MODE_COD	New values have been defined for Automatic Edge Lock mode, Checker mode, and Quality Control.
TEST_COD	Under V3, data type was `C*3`; under V4, data type is `C*n`. The Missing/Invalid flag is now length byte = 0.

MRR: Master Results Record

V3 Fields Dropped From V4:

All part count fields have moved to the PCR (Part Count Record, new in V4):

PART_CNT	RTST_CNT	ABRT_CNT
GOOD_CNT	FUNC_CNT	

PCR: Part Count Record — New in V4

Contains the part counts formerly in the MRR and the SCR. If HEAD_NUM = 255, the counts are for all test sites; otherwise, the counts are for the specified site.

Each STDF file must contain at least one PCR: either one summary PCR (HEAD_NUM = 255), or one PCR for each head/site combination, or both.

Data Fields (after header):

HEAD_NUM	Test head number
SITE_NUM	Test site number
PART_CNT	Number of parts tested
RTST_CNT	Number of parts retested
ABRT_CNT	Number of aborts during testing
GOOD_CNT	Number of good (passed) parts tested
FUNC_CNT	Number of functional parts tested

HBR: Hardware Bin Record

HEAD_NUM and SITE_NUM are added. If HEAD_NUM = 255, the count is for all test sites; otherwise it is for the specified site. Because of these fields, the V3 SHB (Site-Specific Hardware Bin Record) is no longer needed.

The new HBIN_PF field indicates whether the bin was passing or failed.

SBR: Software Bin Record

HEAD_NUM and SITE_NUM are added. If HEAD_NUM = 255, the count is for all test sites; otherwise it is for the specified site. Because of these fields, the V3 SSB (Site-Specific Software Bin Record) is no longer needed.

The new SBIN_PF field indicates whether the bin was passing or failed.

PMR: Pin Map Record

The structure and use of the PMR has changed completely for V4. Under V3, the PMR could define a single channel/pin mapping, or it could define a pin group. Under V4, a PMR defines a single channel/pin mapping. Two more record types have been added — PGR (Pin Group Record) and PLR (Pin List Record) — to define aggregates of pins. See “Using the Pin Mapping Records” on page 77.

V4 Fields:

The PMR is completely redefined under V4. The fields (after the header) are:

PMR_INDX	Unique index associated with pin
CHAN_TYP	Channel type
CHAN_NAM	Channel name
PHY_NAM	Physical name of pin
LOG_NAM	Logical name of pin
HEAD_NUM	Head number associated with channel
SITE_NUM	Site number associated with channel

PGR: Pin Group Record — New in V4

Associates a name with a group of pins.

Data Fields (after header):

GRP_INDX	Unique index associated with pin group
GRP_NAM	Name of pin group
INDX_CNT	Count of PMR indexes
PMR_INDX	Array of indexes for pins in the group

PLR: Pin List Record — New in V4

Defines the current display radix and operating mode for a list of pins or pin groups.

Data Fields (after header):

GRP_CNT	Count of pins or pin groups
GRP_INDX	Array of pin or pin group indexes
GRP_MODE	Operating mode of pin group
GRP_RADX	Display radix of pin group
PGM_CHAR	Program state encoding characters
RTN_CHAR	Return state encoding characters
PGM_CHAL	Program state encoding characters
RTN_CHAL	Return state encoding characters

RDR: Retest Data Record — New in V4

Signals that the data in this STDF file is for retested parts, and indicates what bins are being retested. This data, combined with information in the MIR, tells data filtering programs what data to replace when processing retest data.

If this record is used, it must immediately follow the MIR.

Data Fields (after header):

NUM_BINS	Number of bins being retested
RTST_BIN	Array of retest bin numbers

SDR: Site Description Record — New in V4

A new record type that contains the configuration information for one or more test sites, connected to one test head, that compose a site group. SITE_GRP is a unique identifier for the site group defined by the SDR.

If used, SDRs must immediately follow the MIR and any RDR.

Data Fields (after header):

HEAD_NUM	Test head number
SITE_GRP	Site group number
SITE_CNT	Number of test sites in site group
SITE_NUM	Array of test site numbers
HAND_TYP	Handler or prober type
HAND_ID	Handler or prober ID
CARD_TYP	Probe card type
CARD_ID	Probe card ID
LOAD_TYP	Load board type
LOAD_ID	Load board ID
DIB_TYP	DIB board type
DIB_ID	DIB board ID
CABL_TYP	Interface cable type
CABL_ID	Interface cable ID
CONT_TYP	Handler contactor type
CONT_ID	Handler contactor ID
LASR_TYP	Laser type
LASR_ID	Laser ID
EXTR_TYP	Extra equipment type field
EXTR_ID	Extra equipment ID

WIR: Wafer Information Record

The PAD_BYTE field has been dropped.

The SITE_GRP field has been added, to relate the wafer information to the configuration of the equipment used to test it (as defined in the SDR).

WRR: Wafer Results Record

V3 Fields Dropped from V4:

PAD_BYTE	
HAND_ID	Moved to SDR (identified by SITE_GRP)
PRB_CARD	Moved to SDR (identified by SITE_GRP)

Fields Added for V4:

SITE_GRP	Site group number
FABWF_ID	Fab wafer ID
FRAME_ID	Wafer frame ID
MASK_ID	Wafer mask ID

Other WRR Changes:

These fields have changed from I*4 to U*4: RTST_CNT, ABRT_CNT, GOOD_CNT, and FUNC_CNT. Their Missing/Invalid flag is now 4,294,967,295.

WCR: Wafer Configuration Record

The WF_UNITS field has two new valid values, to indicate that units are in millimeters or in mils. (Previous units were inches and centimeters).

PIR: Part Information Record

Now acts solely as a marker to indicate where testing of a part begins. The fields dropped from V4 are now only in the PRR.

V3 Fields Dropped from V4:

X_COORD	Y_COORD
PART_ID	

PRR: Part Results Record

The PAD_BYTE field has been dropped.

The TEST_T field has been added, for the elapsed test time in milliseconds.

Bits 0 and 1 of PART_FLG now indicate whether the entire sequence of PIR, PTR, MPR, FTR, and PRR records supersedes any previous sequence with the same PART_ID (bit 0) or X & Y coordinates (bit 1). Under V3, this bit meant that only the PIR/PRR pair was superseded.

Bit 4 of PART_FLG is now defined to indicate whether the device completed testing with no pass/fail indication.

PDR: Parametric Test Description Record — Dropped

The PDR has been dropped from V4. In its place, the first PTR for each test will contain the semi-static descriptive information for the test.

FDR: Functional Test Description Record — Dropped

The FDR has been dropped from V4. In its place, the first FTR for each test will contain the semi-static descriptive information for the test.

TSR: Test Synopsis Record

The following fields have been dropped: PAD_BYTE, TST_MEAN, and TST_SDEV.

The data type of the following fields has changed from `I*4` to `U*4`: EXEC_CNT, FAIL_CNT, and ALRM_CNT. The Missing/Invalid flag for these fields is now 4,294,967,295.

HEAD_NUM and SITE_NUM have been added. If HEAD_NUM = 255, the count is for all test sites.

TEST_TYP has been added, to specify the kind of test: parametric, functional, or multiple-result parametric.

TEST_TIM and TEST_LBL have also been added. Bit 2 of OPT_FLAG now indicates that the TEST_TIM value is valid.

PTR: Parametric Test Record

The first PTR for a test establishes the default semi-static descriptive information for that test. This use of the PTR replaces the PDR from V3.

TEST_NAM and SEQ_NAME have been dropped. They are now part of the TSR.

LO_SPEC and HI_SPEC have been added, for low and high spec limit values.

The fields for displaying the parametric test data have changed. The following fields have been dropped:

RES_LDIG	RES_RDIG	DESC_FLG
HLM_LDIG	HLM_RDIG	
LLM_LDIG	LLM_RDIG	

In their place are these fields, which are ANSI C format strings:

C_RESFMT	Test result
C_LLMFMT	Low test and spec limit
C_HLMFMT	High test and spec limit

ALARM_ID has been added.

The data type of UNITS has changed from `C*7` to `C*n`. The Missing/Invalid flag is now length byte = 0.

Bits 6 and 7 of PARM_FLG are now defined, to indicate whether a value that equals the low or high limit is passing or failing.

The following OPT_FLAG bits have changed:

bit 1:	Reserved for future use
bit 2:	No low specification limit
bit 3:	No high specification limit

MPR: Multiple-Result Parametric Record — New in V4

Contains the results of a single execution of a parametric test in the test program where that test returns multiple values.

The first MPR for a test establishes the default semi-static descriptive information for that test.

Data Fields (after header):

TEST_NUM	Test number
HEAD_NUM	Test head number
SITE_NUM	Test site number
TEST_FLG	Test flags (fail, alarm, etc.)
PARM_FLG	Parametric test flags (drift, etc.)
RTN_ICNT	Count of PMR indexes
RSLT_CNT	Count of returned results
RTN_STAT	Array of returned states
RTN_RSLT	Array of returned results
TEST_TXT	Descriptive text or label
ALARM_ID	Name of alarm
OPT_FLAG	Optional data flag
RES_SCAL	Test result scaling exponent
LLM_SCAL	Test low limit scaling exponent
HLM_SCAL	Test high limit scaling exponent
LO_LIMIT	Test low limit value
HI_LIMIT	Test high limit value
START_IN	Starting input value (condition)
INCR_IN	Increment of input condition
RTN_INDX	Array of PMR indexes
UNITS	Units of returned results
UNITS_IN	Input condition units
C_RESFMT	ANSI C result format string
C_LLMFMT	ANSI C low limit format string
C_HLMFMT	ANSI C high limit format string
LO_SPEC	Low specification limit value
HI_SPEC	High specification limit value

FTR: Functional Test Record

The FTR has been significantly restructured for V4. The lists below show what fields have been dropped and added.

Fields Dropped From V4:

DESC_FLG	
VECT_ADR	(compare the V4 field REL_VADR)
PCP_ADDR	
VECT_DAT	
DEV_DAT	
RPIN_MAP	(compare the V4 field SPIN_MAP)
TEST_NAM	(moved to the TSR for this test)
SEQ_NAME	(moved to the TSR for this test)

Fields Added for V4:

REL_VADR	Relative vector address
XFAIL_AD	X logical device failure address
YFAIL_AD	Y logical device failure address
VECT_OFF	Offset from vector of interest
RTN_ICNT	Count of return data PMR indexes
PGM_ICNT	Count of programmed state indexes
RTN_INDX	Array of return data PMR indexes
RTN_STAT	Array of returned states
PGM_INDX	Array of programmed state indexes
PGM_STAT	Array of programmed states
VECT_NAM	Vector module pattern name
OP_CODE	Vector Op Code
ALARM_ID	Name of alarm
PROG_TXT	Additional programmed information
RSLT_TXT	Additional result information
PATG_NUM	Pattern generator number
SPIN_MAP	Bit map of enabled comparators

Other FTR Changes:

The first FTR for a test establishes the default semi-static descriptive information for that test. This use of the FTR replaces the FDR from V3. Specifically, the fields PATG_NUM and SPIN_MAP (both new with V4) contain semi-static information.

These data types have changed:

- REPT_CNT has changed from `U*2` to `U*4`. Its Missing/Invalid flag is OPT_FLAG bit 2 =1.
- FAIL_PIN has changed form `B*n` to `D*n`.
- TIME_SET has changed from `U*1` to `C*n`. Its Missing/Invalid flag is length byte = 0.

(continued)

Main Menu

Other FTR Changes (continued):

The meanings of the bit settings for OPT_FLAG have completely changed. Consult the *STDF Specification*.

The meaning of Bit 1 of TEST_FLG has changed. It no longer indicates channel vs. pin. It is now reserved for future use.

SHB: Site-Specific Hardware Bin Record — Dropped

The functionality of the SHB has been incorporated into the HBR.

SSB: Site-Specific Software Bin Record — Dropped

The functionality of the SSB has been incorporated into the SBR.

STS: Site-Specific Test Synopsis Record — Dropped

The functionality of the STS has been incorporated into the TSR.

SCR: Site-Specific Part Count Record — Dropped

The functionality of the SCR has been incorporated into the PCR (new with V4).

The following records are **unchanged** between Version 3 and Version 4:

BPS:	Begin Program Section Record
EPS:	End Program Section Record
GDR:	Generic Data Record
DTR:	Datalog Text Record

Glossary

Aborted part

A part is considered to have aborted if testing began on the part, but the part was not tested to completion. For example, the operator may have interrupted testing of the part via a keyboard command.

ADART

(Automatic Distribution Analysis in Real Time) A program used to perform statistical analysis of test results in the testing computer. ADART produces histograms or cumulative plots of test data, which may be read at any time during the testing process.

ASCII

(American Standard Code for Information Interchange) A code, using seven bit plus parity, established by the American National Standards Institute (ANSI) to achieve compatibility between devices exchanging character oriented data.

Data base

An electronic organization of data and information organized and maintained by a data base management system. Data base implies integration of data across the entire environment that it serves. It also implies central control of data for consistency and accuracy with users having access to their authorized view of it.

Datalog

Listing of specific test information, such as test results and parameter values.

Die

A single semiconductor device within a wafer.

Executive

A program or set of programs that provides a user environment to testing, program development, debugging, data analysis services for a tester. Also known as a MOP.

Field

A defined unit of data/information in a record. A field defines the physical storage location of a unit of data/information. One or more fields make up a record. A group of records is called a file.

File

A group of related data elements (records) arranged in a structure significant to the user and usually treated as a unit. A file can contain data, programs, or both.

File specification

A name that uniquely identifies a file maintained in any operating system. A file specification generally consists of up to six components: (1) a node name specifying which computer in the network owns the data; (2) a device name identifying the volume on which the file is stored; (3) a directory name indicating the logical path for accessing the file on the volume; (4) a file name; (5) a file extension; and (6) a file version number. Not all operating systems support the full set of six components.

Finish time

The time at which the last device in the lot is finished testing.

Functional part

Any part that, when tested, does not go into the catastrophic failure bin (usually bin 0). The count of functional parts is kept in the FUNC_CNT field of the MRR and WRR and is necessary for calculating the good-to-functional ratio.

Good part

Any part that, when tested, is placed in a bin containing parts acceptable for use and/or sale. The count of good parts is kept in the GOOD_CNT field of the MRR and WRR and is necessary for calculating the yield and good-to-functional ratio.

Hardware

Physical equipment as opposed to a computer program or method of use.

Hardware bins

Physical sort categories connected with a device handler for grading tested devices.

Histogram

A graphic representation of a frequency distribution in which the widths of the contiguous vertical bars are proportional to the class intervals of the variables, and the heights of the bars are proportional to the number of times that statistical data had a value that fell into a class interval.

Host computer

A computer attached to a network providing centralized primary services such as data base access, data analysis software, test floor monitoring, test floor control, and program development tools.

Insertion

The act of testing one lot of parts one time. A lot of parts may be tested several times under different test conditions (such as wafer, cold, hot, pre-burnin, post-burnin, etc.).

Job plan

A set of related program statements grouped together in modules, designed to test a specific part or device. Test engineers write, edit, and compile job plans on the testers, at work stations, or on the host computer. Job Plans are also known as test plans or test programs.

Lot

A batch of parts (often an entire production run) to be tested as a group through one or more test cycles. A lot may be tested as a whole or as sublots. A lot may consist of devices, boards, or wafers in quantities from one to thousands.

Lot disposition

A lot disposition is a decision as to the future of the lot. For example, after testing a lot of wafers it may be decided that the yield was so low that the devices should not be packaged.

Lot disposition code

A character code indicating the lot disposition.

Master operating program (MOP)

A program that functions as an operating system in a tester. More generally, a MOP is any stand-alone program which can be bootstrapped into a network node. Also known as an executive.

Network

An interconnected group of computers linked together for specific purposes, such as sharing data files. ATE networks generally include tester computers, test plan development stations, and host computers.

Network architecture

A formalized definition of the structures and interactions required to provide shared communications functions.

Node

Any intelligent device that is connected to a network and is capable of sending and receiving network messages.

Operating system

Software that controls the execution of computer programs and provides some or all of the following services: scheduling, debugging, input and output control, accounting, storage assignment, and data management. Examples of operating systems include VMS, UNIX, RSX-11, and VM/CMS.

Operator

A person responsible for testing parts at one test station of a tester.

Parts

For the purpose of this document, parts are electronic devices (both discrete components and integrated circuits) and PC boards.

Privilege

A characteristic of a user or program that determines what kinds of operations a user or a program can perform. In general, a privileged user or program can affect system operations and/or data.

Retested part

A part which was tested more than once during the course of one insertion of a lot is called a retest. Usually, parts will only be retested if a problem was detected the first time the part was tested. For example, a part may be retested if it was inserted upside down or in the handler contacts were not functioning properly

Sequencer

A sequencer (or sequencer function) can be viewed as the table of contents of a test program. The sequencer function is a list of all the tests to be performed in order of their execution. For each test, all limits, datalog formatting information, and binning information is presented in a tabular, readable form, resembling a specification sheet.

Setup time

The time at which the operator begins setting up the tester for testing a lot. Setup includes loading the job, adjusting the handler or prober, setting up the test head, setting up datalog parameters, and any other operations which must be performed before the first part is tested.

Software

A set of computer programs, procedures, rules, and associated documentation concerned with the operation of computer systems.

Software bins

Logical sort categories implemented in the test plan for finer categorization of tested parts than is provided by the hardware bins on the device handler. Software bins are often used to detect degrees of “goodness” of devices so that the effect of variations in the fabrication process can be more accurately predicted.

Start time

The time at which the first device in the lot (or wafer) begins testing.

Sublot

A portion of a full lot of parts to be tested. Lots are often divided into sublots to facilitate handling or tester scheduling.

Tester

A machine capable of separating good parts from bad. Most device testers are capable of grading parts as well. All but the simplest testers are built with one or more computers and are capable of test data collection and networking.

Test data

Raw and derived information collected from parts measured by a tester. Test data is used for measuring the “goodness” of the parts being tested and of the process used in making those parts.

Test head

A test head is a physical entity consisting of the hardware connections necessary to test one or more devices. On parallel testers, a test head controls multiple test sites; on non-parallel testers, test heads and test sites are equivalent. Each tester supports one or more test heads capable of testing parts.

Test plan

A set of related program statements grouped together in modules, designed to test a specific part or device. Test engineers write, edit, and compile test plans on the testers, at work stations, or on the host computer. Test Plans are also known as job plans or test programs.

Test program

See test plan.

Test site

A test site consists of the hardware connections necessary to test a single device. There may be one or more test sites associated with a test head.

Test station

A test station is a logical software entity capable of loading and running a single test plan. When used for testing parts, a test station is associated with one or more test heads. In some testers each test station is permanently assigned to a single test head, while in others the assignment is created by a software command. Each tester has one or more test stations capable of executing test plans.

Wafer

A disk of single-crystal, high-purity semiconducting material used as the substrate in the manufacture of integrated circuits. Wafers are processed in a series of steps which add or subtract materials of a controlled size, shape, and purity to create integrated circuits. Each wafer is then probed by Automatic Test Equipment. Good devices, or dice, are then assembled into packages for final testing.

STDF Specification V4

Index

Click on any page number.

A

B

D

E

F

G