The Summary File Format for `SDDFStatistics` Revision 1.6

Dave Kohr

April 21, 1994

Abstract

The `SDDFStatistics` program gathers a variety of statistics about the data fields of SDDF files, and can save these statistics to summary files which are themselves in SDDF format. These summary files are used by `SDDFStatistics` as "caches" of summary data, to minimize startup time when interactively browsing SDDF data files using the graphical user interface provided by `SDDFStatistics`. They may also be used by other Pablo tools, including the main Pablo Visualization Environment, whose displays may be configured more easily using information contained in the summaries. This document defines and describes the format of the summary files produced by `SDDFStatistics`, and includes an example consisting of an input SDDF data file and the corresponding summary file generated by `SDDFStatistics`.

1 Overview

`SDDFStatistics` computes statistics for a single SDDF file at the following four levels of detail:

1. For the entire file.
2. For each record type in the file.
3. For each field of each record type.
4. For each dimension of each array field of each record.

Statistics at level one (the entire file) are stored in a single SDDF record of type `"File Summary"`. In addition, for each record type named *RecordName* in the original SDDF input file, there is a corresponding summary record type named "*RecordName*`::_summary_`" in the output summary file. Statistics for levels two through four appear in these latter summary records.

Finally, `SDDFStatistics` inserts a single attribute into the summary file with key `"generated by"`, which applies to the entire output stream. The value of this attribute is the name and version number of the program that generated the summary file.

2 The `"File Summary"` Record Type

At the highest level of detail, there will be a single instance of a record of type `"File Summary"`, whose structure is defined in Figure 1. Note that for each field, there is an attribute whose key is `"description"` and whose value is a brief description of the contents of the field. Likewise, there is a `"description"` attribute which describes the contents of the entire record.

2.1 The "name" and "size" Fields

The "name" field is an ASCII string for the relative Unix pathname from the directory from which SDDFStatistics was invoked to the input SDDF file. "size" contains the size of the input file in bytes.

2.2 The SDDF Packet Count Fields

The "descriptor count", "data count", "attribute count", and "command count" fields contain the number of SDDF packets in the input file for data record descriptors, data records, stream attributes, and commands, respectively. The "duplicate descriptor count" field contains the number of data record descriptors in the input file whose tag numbers were duplicates of the tag of some previously-appearing record descriptor.

2.3 The "dominant type" Field

The final field, "dominant type", contains a summary of which record types were most prevalent in various portions of the input file. If the "dominant type" array has N elements, then the input file is considered to consist of N contiguous non-overlapping sections, such that each contains an equal number of data records (or very nearly equal, if the number of data records in the input file is not evenly divisible by N), and such that all of the sections taken together span the entire input file. Suppose that in section zero of the file, the record type *RecordName*, with tag t, is most common. Then in the summary file, there will be a corresponding record type "*RecordName*::_summary_", as explained in §1. This summary record will have some tag τ, which may not be identical to t. Element zero of "dominant type" will be τ. Similarly, if in section one of the file, the most prevalent record type has tag u, and the tag of its corresponding summary record is v, then element one of "dominant type" will be v, and so on up to section $N-1$ of the input file. Within a given section of the input file, if two or more record types are tied for the highest frequency of occurrence, then SDDFStatistics will choose one of them arbitrarily as being "most prevalent" in that section.

3 Summary Records for Each Input Record Type

As stated in §1, for each record type named *RecordName* in the original SDDF input file, there is a corresponding summary record type named "*RecordName*::_summary_" in the output summary file. (These summary records are also referred to as "per-record summary records" in this document.) Such summary record types contain two kinds of fields:

1. "Fixed" fields containing information pertaining to the entire input record type (Level two of the data classification presented in §1).

2. "Per-field" fields containing information pertaining to just one of the fields in the input record type (Levels three and four of the classification).

Figure 2 shows a portion of the structure of the summary record for an example input record named "RecordName", whose first field is named "FirstFieldName".

In the figure, the scalar base type of "FirstFieldName" is denoted by "FirstFieldType", which may be any of the SDDF scalar base types "char", "int", "float", or "double". "FirstFieldName" may be either an array of elements of type "FirstFieldType", or a single scalar value of that type; the structure would be the same in either case. Each summary record field has an attribute with key "description"

```
// "description" "Summary data for entire file"
"File Summary" {
        // "description" "original SDDF file name"
        char    "name"[];
        // "description" "file size in bytes"
        int     "size";
        // "description" "total number of Descriptor packets"
        int     "descriptor count";
        // "description" "total number of Data packets"
        int     "data count";
        // "description" "total number of Attribute packets"
        int     "attribute count";
        // "description" "total number of Command packets"
        int     "command count";
        // "description" "number of Descriptor packets with duplicate tags"
        int     "duplicate descriptor count";
        // "description" "tag of dominant record type in each fraction of file"
        int     "dominant type"[];
};;
```

Figure 1: Structure of the `"File Summary"` record type.

whose value is a brief description of the contents of that field. Note that for compactness, only the first set of "per-field" fields in the summary record descriptor, corresponding to the first field of the input record, has a `"description"` attribute.

The SDDF descriptor packets for summary record types appear in the same relative order in the summary file as did the corresponding descriptor packets for input record types in the original input file. This facilitates implementation of the `SDDFStatistics` option for sorting record types by the order of appearance of their descriptors in the input file.

3.1 The Fixed Fields

The two "fixed" fields always appear at the beginning of a summary record. The `"__record count__"` field contains the number of instances in the input file of a data record of the input record type corresponding to this summary record. The `"__record tag__"` field contains the SDDF record tag of the input record type (which is not necessarily the same as the summary record tag, as mentioned above). Note that the name of the input record type may be computed from the name of its corresponding summary record type.[1]

3.2 The Per-field Fields

There is one repetition of the sequence of twelve "per-field" fields in the summary record for each field in the input record. If the input field name is *FieldName*, each of the twelve resulting summary field names has the prefix "*FieldName::*". The rest of the summary field name denotes one of the twelve kinds of "per-field" summary fields; hence the summary fields are named "*FieldName*`::__min__`", "*FieldName*`::__max__`", and

[1] There is a function for doing so in the programming interface discussed in §5.

```
// "description" "Summary data for \"RecordName\" record"
"RecordName::__summary__" {
        // "description" "number of records of this type in file"
        int             "__record count__";
        // "description" "record type-tag in original input file"
        int             "__record tag__";
        // "description" "minimum value in entire file"
        FirstFieldType  "FirstFieldName::__min__";
        // "description" "maximum value in entire file"
        FirstFieldType  "FirstFieldName::__max__";
        // "description" "count of individual scalar values for this field"
        int             "FirstFieldName::__values count__";
        // "description" "average value in entire file"
        double          "FirstFieldName::__mean__";
        // "description" "standard deviation of all values in entire file"
        double          "FirstFieldName::__stddev__";
        // "description" "low cutoff value for histogram"
        FirstFieldType  "FirstFieldName::__underflow__";
        // "description" "high cutoff value for histogram"
        FirstFieldType  "FirstFieldName::__overflow__";
        // "description" "number of bins in histogram"
        int             "FirstFieldName::__bin count__";
        // "description" "count of elements in each bin of histogram"
        int             "FirstFieldName::__histogram__"[];
        // "description" "minimum size of each dimension"
        int             "FirstFieldName::__min dimension__"[];
        // "description" "maximum size of each dimension"
        int             "FirstFieldName::__max dimension__"[];
        // "description" "average size of each dimension"
        double          "FirstFieldName::__mean dimension__"[];
        // "description" "standard deviation of size of each dimension"
        double          "FirstFieldName::__stddev dimension__"[];
        .
        .
        .
};;
```

Figure 2: A portion of the structure of an example summary record.

so on up to "*FieldName*::__stddev dimension__". The discussion below of these twelve kinds of summary fields assumes the input field name is *FieldName*, of scalar base type *FieldType*.

3.2.1 The "min" and "max" Fields

The fields "*FieldName*::__min__" and "*FieldName*::__max__" contain the minimum and maximum of all values for field *FieldName* in all instances of its record type in the input file. If *FieldType* is an array type, then each element of each instance of *FieldName* is examined to compute the minimum and maximum (henceforth, this is called "expanding" the array). Note also that the types of "*FieldName*::__min__" and "*FieldName*::__max__" are *FieldType* rather than a fixed SDDF scalar base type.

3.2.2 The "values count", "mean", and "stddev" Fields

The field "*FieldName*::__values count__" contains a count of all values for field *FieldName* in all instances of its record type in the input file. If *FieldName* is of a scalar type, then each instance counts once. If *FieldName* is of an array type, then each instance is "expanded," so that each element of an instance counts once.

"*FieldName*::__mean__" contains the arithmetic mean of all values in instances for *FieldName*, with arrays expanded. "*FieldName*::__stddev__" contains the "population" standard deviation σ of all values, where

$$\sigma^2 = \frac{\sum_{v \in V} v^2}{N} - (\overline{V})^2,$$

V is the set of N values occurring in instances of the field (with arrays expanded), and $\overline{V}$ is the mean of the values in V (identical to "*FieldName*::__mean__").

3.2.3 The Histogram Fields

The next three fields represent a histogram of values in instances of *FieldName*, with arrays expanded. The fields "*FieldName*::__underflow__" and "*FieldName*::__overflow__" are the minimum and maximum values, respectively, which are present in the histogram: values below the minimum or above the maximum are not used in computing the histogram. By selecting "*FieldName*::__underflow__" to be greater than "*FieldName*::__min__", or "*FieldName*::__overflow__" to be less than "*FieldName*::__max__", one may cause the histogram to skip outliers and represent only significant data points.

"*FieldName*::__histogram__" is a 1-dimensional array of integers which contains the histogram itself. If "*FieldName*::__histogram__" contains N elements, then there are N "bins" in the histogram. Abbreviating "*FieldName*::__underflow__" by l ("low") and "*FieldName*::__overflow__" by h ("high"), each bin has a width of $w = (h - l)/N$. Element i of "*FieldName*::__histogram__" contains a count of the number of values falling into bin i of the histogram. For $0 \leq i < N - 1$, bin i represents values in the open interval

$$[l + w \times i,\ l + w \times (i + 1))$$

Bin $N - 1$ represents values in the closed interval

$$[l + w \times (N - 1),\ h]$$

3.2.4 The Dimension Statistics Fields

Fields "*FieldName*::`_min dimension_`" through "*FieldName*::`_stddev dimension_`" contain statistics about the dimensions of instances of *FieldName* in the input file. Each is a 1-dimensional array. If *FieldType* is an array type with dimension N, then each of these summary fields has N elements. This is also true for the case when *FieldType* is actually a scalar type: then $N = 0$, and the summary fields are 1-dimensional arrays containing zero elements. Therefore, a simple way to determine the dimensionality of the input field `"FieldName"` is to look up the number of elements contained in any of its 4 corresponding dimension statistics fields.

For each of these summary fields, element i represents a statistic for dimension i, with $0 \leq i < N$ (the first dimension is numbered as zero). "*FieldName*::`_min dimension_`" and "*FieldName*::`_max dimension_`" hold the minimum and maximum size of each dimension among all instances of *FieldName* in the input file. "*FieldName*::`_mean dimension_`" contains the mean size of each dimension, and "*FieldName*::`_stddev dimension_`" contains the "population" standard deviation of the size of each dimension.

4 A Complete Example

Figure 3 contains a sample input SDDF file, and Figures 4 through 9 are the corresponding summary file produced by `SDDFStatistics`. This example illustrates most of the important features of summary files, though for brevity the input file has been kept unrealistically small. Note that although the SDDF Library inserts whitespace into its output files differently from these typeset depictions, the examples are still perfectly valid as input files to the SDDF Library.[2]

The record types in the input file highlight the following characteristics of `SDDFStatistics`:

- The `"Test Scalar Types"` record type shows the summary output produced for scalar fields of each of the four SDDF scalar base types `"char"`, `"int"`, `"float"`, and `"double"`.
- The `"Test Array Types"` record type shows the summary output produced for 1-dimensional array fields of each of the 4 scalar base types, as well as for a 2-dimensional array.
- The `"Absent Record"` record type shows the summary output produced for scalar and 1-dimensional array fields of base types `"char"` and `"int"` when no instances of the record appear in the input.

The total size of the summary data record instances in the summary file remains fixed regardless of the number of data instances in the input file. Therefore, in a more realistic case with a large number of input data instances, the summary will typically be much smaller than the input (unlike the sample input and summary files presented here). For example, during its development `SDDFStatistics` was tested with a 3,500,543 byte binary file of trace data captured using the Pablo Instrumentation Library; the corresponding summary file was 95,060 bytes. It is expected that most Pablo Instrumentation trace file summaries will be about this size.

5 The Programming Interface for Manipulating Summary Data

A very rudimentary C++ programming interface exists for manipulating `SDDFStatistics` summary data. All of the summary data for a single input file is contained in an object of class `PerFileSummary`. The

[2] In the figures, the original SDDF Library ASCII format has been edited by hand to improve readability and reduce use of vertical space.

summary data is represented internally within `PerFileSummary` using a single SDDF `RecordDictionary` which contains one `RecordDossier` for the `"File Summary"` summary record (whose structure is defined in Figure 1), and one `RecordDossier` for each of the "per-record" summary records (whose structure is of the form presented in Figure 2).

The programming interface provides the following operations on `PerFileSummary` objects:

- An object may be created from an input SDDF file (so that computation of the summary data occurs during initialization of the object).
- An object may be created from an existing summary file (so that no summary data computation is performed).
- The contents of an object may be saved to a summary file.
- Read-only access to the summary data `RecordDictionary` of an existing object may be obtained using an accessor function.
- Using methods provided by the `RecordDictionary` class, access may be obtained to `RecordDossier` objects for summary data records.
- The fields of the `"File Summary"` summary record may be examined using a variety of translation function and data members provided by class `PerFileSummary` (in conjunction with SDDF Library functions for the classes `RecordDossier`, `Array`, `Value`, and others).
- The fields of "per-record" summary records may be examined using a variety of translation function and data members provided by class `PerRecordSummary` (in conjunction with SDDF Library functions for the classes `RecordDossier`, `Array`, `Value`, and others).

The C++ source code for the programming interface is contained in the files `SDDFStats_Misc.C`, `SDDFStats_Misc.h`, `SummaryInfo.C`, and `SummaryInfo.h` within the source directory for `SDDFStatistics`. It is intended that these files be compiled separately from the rest of the `SDDFStatistics` source, relying only on the contents of the SDDF Library header files, so that the summary data interface they provide can be re-used in other Pablo programs.

For more details on this interface, see header file `SummaryInfo.h`. It declares all the available interface functions and data definitions. The rest of the `SDDFStatistics` source code contains examples of using the interface.

```
SDDFA
#1:
// "description" "Test all scalar data types"
"Test Scalar Types" {
        // "aChar" "Test char type"
        char    "aChar";
        // "anInt" "Test int type"
        int     "anInt";
        // "aFloat" "Test float type"
        float   "aFloat";
        // "aDouble" "Test double type"
        double  "aDouble";
};;
#2:
// "description" "Test all array types"
"Test Array Types" {
        // "charArray" "Test char array"
        char    "charArray"[];
        // "intArray" "Test int array"
        int     "intArray"[];
        // "floatArray" "Test float array"
        float   "floatArray"[];
        // "doubleArray" "Test double array"
        double  "doubleArray"[];
        // "int2DArray" "Test 2-D int array"
        int     "int2DArray"[][];
};;
#3:
// "description" "Test default summary field values"
"Absent Record" {
        char    "aChar";
        char    "someChars"[];
        int     "anInt";
        int     "someInts"[];
};;

"Test Scalar Types" { 1, 1, 1, 1 };;
"Test Scalar Types" { 4, 4, 4, 4 };;
"Test Scalar Types" { 2, 2, 2, 2 };;
"Test Array Types" {
        [3] { "abc" },
        [0] { },
        [3] { 1.0, -2.0, 3.0 },
        [2] { 10.0, -10.0 },
        [1][2] { 123, 456 } };;
"Test Array Types" {
        [2] { "de" },
        [2] { -23, 24 },
        [1] { 123.0 },
        [2] { 10.0, -10.0 },
        [0][0] { } };;
"Test Array Types" {
        [4] { "fghi" },
        [1] { 12 },
        [2] { 123.0, -123.0 },
        [2] { 1.0, -1.0 },
        [3][1] { 1, 2, 3 } };;
```

Figure 3: A sample input SDDF file.

```
SDDFA
/*
 * "generated by" "SDDFStatistics version 0.1"
 */ ;;
#100000:
// "description" "Summary data for entire file"
"File Summary" {
        // "description" "original SDDF file name"
        char    "name"[];
        // "description" "file size in bytes"
        int     "size";
        // "description" "total number of Descriptor packets"
        int     "descriptor count";
        // "description" "total number of Data packets"
        int     "data count";
        // "description" "total number of Attribute packets"
        int     "attribute count";
        // "description" "total number of Command packets"
        int     "command count";
        // "description" "number of Descriptor packets with duplicate tags"
        int     "duplicate descriptor count";
        // "description" "tag of dominant record type in each fraction of file"
        int     "dominant type"[];
};;
#1:
// "description" "Summary data for \"Test Scalar Types\" record"
"Test Scalar Types::__summary__" {
        // "description" "number of records of this type in file"
        int     "__record count__";
        // "description" "record type-tag in original input file"
        int     "__record tag__";
        // "description" "minimum value in entire file"
        char    "aChar::__min__";
        // "description" "maximum value in entire file"
        char    "aChar::__max__";
        // "description" "count of individual scalar values for this field"
        int     "aChar::__values count__";
        // "description" "average value in entire file"
        double  "aChar::__mean__";
        // "description" "standard deviation of all values in entire file"
        double  "aChar::__stddev__";
        // "description" "low cutoff value for histogram"
        char    "aChar::__underflow__";
        // "description" "high cutoff value for histogram"
        char    "aChar::__overflow__";
        // "description" "count of elements in each bin of histogram"
        int     "aChar::__histogram__"[];
        // "description" "minimum size of each dimension"
        int     "aChar::__min dimension__"[];
        // "description" "maximum size of each dimension"
        int     "aChar::__max dimension__"[];
        // "description" "average size of each dimension"
        double  "aChar::__mean dimension__"[];
        // "description" "standard deviation of size of each dimension"
        double  "aChar::__stddev dimension__"[];
        int     "anInt::__min__";
        int     "anInt::__max__";
        int     "anInt::__values count__";
        double  "anInt::__mean__";
        double  "anInt::__stddev__";
```

Figure 4: A sample summary file produced by `SDDFStatistics`.

```
        int     "anInt::__underflow__";
        int     "anInt::__overflow__";
        int     "anInt::__histogram__"[];
        int     "anInt::__min dimension__"[];
        int     "anInt::__max dimension__"[];
        double  "anInt::__mean dimension__"[];
        double  "anInt::__stddev dimension__"[];
        float   "aFloat::__min__";
        float   "aFloat::__max__";
        int     "aFloat::__values count__";
        double  "aFloat::__mean__";
        double  "aFloat::__stddev__";
        float   "aFloat::__underflow__";
        float   "aFloat::__overflow__";
        int     "aFloat::__histogram__"[];
        int     "aFloat::__min dimension__"[];
        int     "aFloat::__max dimension__"[];
        double  "aFloat::__mean dimension__"[];
        double  "aFloat::__stddev dimension__"[];
        double  "aDouble::__min__";
        double  "aDouble::__max__";
        int     "aDouble::__values count__";
        double  "aDouble::__mean__";
        double  "aDouble::__stddev__";
        double  "aDouble::__underflow__";
        double  "aDouble::__overflow__";
        int     "aDouble::__histogram__"[];
        int     "aDouble::__min dimension__"[];
        int     "aDouble::__max dimension__"[];
        double  "aDouble::__mean dimension__"[];
        double  "aDouble::__stddev dimension__"[];
};;
#2:
// "description" "Summary data for \"Test Array Types\" record"
"Test Array Types::__summary__" {
        // "description" "number of records of this type in file"
        int     "__record count__";
        // "description" "record type-tag in original input file"
        int     "__record tag__";
        // "description" "minimum value in entire file"
        char    "charArray::__min__";
        // "description" "maximum value in entire file"
        char    "charArray::__max__";
        // "description" "count of individual scalar values for this field"
        int     "charArray::__values count__";
        // "description" "average value in entire file"
        double  "charArray::__mean__";
        // "description" "standard deviation of all values in entire file"
        double  "charArray::__stddev__";
        // "description" "low cutoff value for histogram"
        char    "charArray::__underflow__";
        // "description" "high cutoff value for histogram"
        char    "charArray::__overflow__";
        // "description" "count of elements in each bin of histogram"
        int     "charArray::__histogram__"[];
        // "description" "minimum size of each dimension"
        int     "charArray::__min dimension__"[];
        // "description" "maximum size of each dimension"
        int     "charArray::__max dimension__"[];
        // "description" "average size of each dimension"
```

Figure 5: A sample summary file (continued).

```
        double  "charArray::__mean dimension__"[];
        // "description" "standard deviation of size of each dimension"
        double  "charArray::__stddev dimension__"[];
        int     "intArray::__min__";
        int     "intArray::__max__";
        int     "intArray::__values count__";
        double  "intArray::__mean__";
        double  "intArray::__stddev__";
        int     "intArray::__underflow__";
        int     "intArray::__overflow__";
        int     "intArray::__histogram__"[];
        int     "intArray::__min dimension__"[];
        int     "intArray::__max dimension__"[];
        double  "intArray::__mean dimension__"[];
        double  "intArray::__stddev dimension__"[];
        float   "floatArray::__min__";
        float   "floatArray::__max__";
        int     "floatArray::__values count__";
        double  "floatArray::__mean__";
        double  "floatArray::__stddev__";
        float   "floatArray::__underflow__";
        float   "floatArray::__overflow__";
        int     "floatArray::__histogram__"[];
        int     "floatArray::__min dimension__"[];
        int     "floatArray::__max dimension__"[];
        double  "floatArray::__mean dimension__"[];
        double  "floatArray::__stddev dimension__"[];
        double  "doubleArray::__min__";
        double  "doubleArray::__max__";
        int     "doubleArray::__values count__";
        double  "doubleArray::__mean__";
        double  "doubleArray::__stddev__";
        double  "doubleArray::__underflow__";
        double  "doubleArray::__overflow__";
        int     "doubleArray::__histogram__"[];
        int     "doubleArray::__min dimension__"[];
        int     "doubleArray::__max dimension__"[];
        double  "doubleArray::__mean dimension__"[];
        double  "doubleArray::__stddev dimension__"[];
        int     "int2DArray::__min__";
        int     "int2DArray::__max__";
        int     "int2DArray::__values count__";
        double  "int2DArray::__mean__";
        double  "int2DArray::__stddev__";
        int     "int2DArray::__underflow__";
        int     "int2DArray::__overflow__";
        int     "int2DArray::__histogram__"[];
        int     "int2DArray::__min dimension__"[];
        int     "int2DArray::__max dimension__"[];
        double  "int2DArray::__mean dimension__"[];
        double  "int2DArray::__stddev dimension__"[];
};;
#3:
// "description" "Summary data for \"Absent Record\" record"
"Absent Record::__summary__" {
        // "description" "number of records of this type in file"
        int     "__record count__";
        // "description" "record type-tag in original input file"
        int     "__record tag__";
```

Figure 6: A sample summary file (continued).

```
        // "description" "minimum value in entire file"
        char    "aChar::__min__";
        // "description" "maximum value in entire file"
        char    "aChar::__max__";
        // "description" "count of individual scalar values for this field"
        int     "aChar::__values count__";
        // "description" "average value in entire file"
        double  "aChar::__mean__";
        // "description" "standard deviation of all values in entire file"
        double  "aChar::__stddev__";
        // "description" "low cutoff value for histogram"
        char    "aChar::__underflow__";
        // "description" "high cutoff value for histogram"
        char    "aChar::__overflow__";
        // "description" "count of elements in each bin of histogram"
        int     "aChar::__histogram__"[];
        // "description" "minimum size of each dimension"
        int     "aChar::__min dimension__"[];
        // "description" "maximum size of each dimension"
        int     "aChar::__max dimension__"[];
        // "description" "average size of each dimension"
        double  "aChar::__mean dimension__"[];
        // "description" "standard deviation of size of each dimension"
        double  "aChar::__stddev dimension__"[];
        char    "someChars::__min__";
        char    "someChars::__max__";
        int     "someChars::__values count__";
        double  "someChars::__mean__";
        double  "someChars::__stddev__";
        char    "someChars::__underflow__";
        char    "someChars::__overflow__";
        int     "someChars::__histogram__"[];
        int     "someChars::__min dimension__"[];
        int     "someChars::__max dimension__"[];
        double  "someChars::__mean dimension__"[];
        double  "someChars::__stddev dimension__"[];
        int     "anInt::__min__";
        int     "anInt::__max__";
        int     "anInt::__values count__";
        double  "anInt::__mean__";
        double  "anInt::__stddev__";
        int     "anInt::__underflow__";
        int     "anInt::__overflow__";
        int     "anInt::__histogram__"[];
        int     "anInt::__min dimension__"[];
        int     "anInt::__max dimension__"[];
        double  "anInt::__mean dimension__"[];
        double  "anInt::__stddev dimension__"[];
        int     "someInts::__min__";
        int     "someInts::__max__";
        int     "someInts::__values count__";
        double  "someInts::__mean__";
        double  "someInts::__stddev__";
        int     "someInts::__underflow__";
        int     "someInts::__overflow__";
        int     "someInts::__histogram__"[];
        int     "someInts::__min dimension__"[];
        int     "someInts::__max dimension__"[];
        double  "someInts::__mean dimension__"[];
        double  "someInts::__stddev dimension__"[];
};;
```

Figure 7: A sample summary file (continued).

```
"File Summary" {
        [48] { "/home/reed/drk/Pablo/FileStats/Src/TestData.txt" },
        1281, 3, 6, 0, 0, 0,
        [6] { 1, 1, 1, 2, 2, 2 } };;

"Test Scalar Types::__summary__" {
        3, 1, 1, 4, 3, 50.33333333333334, 1.247219128924505, 1, 4,
        [10] { 1, 0, 0, 1, 0, 0, 0, 0, 0, 1 },
        [0] { },
        [0] { },
        [0] { },
        [0] { },
        1, 4, 3, 2.333333333333333, 1.247219128924647, 1, 4,
        [10] { 1, 0, 0, 1, 0, 0, 0, 0, 0, 1 },
        [0] { },
        [0] { },
        [0] { },
        [0] { },
        1, 4, 3, 2.333333333333333, 1.247219128924647, 1, 4,
        [10] { 1, 0, 0, 1, 0, 0, 0, 0, 0, 1 },
        [0] { },
        [0] { },
        [0] { },
        [0] { },
        1, 4, 3, 2.333333333333333, 1.247219128924647, 1, 4,
        [10] { 1, 0, 0, 1, 0, 0, 0, 0, 0, 1 },
        [0] { },
        [0] { },
        [0] { },
        [0] { } };;
"Test Array Types::__summary__" {
        3, 2, a, i, 9, 101, 2.581988897471494, a, i,
        [10] { 1, 1, 1, 1, 0, 1, 1, 1, 1, 1 },
        [1] { 2 },
        [1] { 4 },
        [1] { 3 },
        [1] { 0.8164965809277257 },
        -23, 24, 3, 4.333333333333333, 19.93879523831757, -23, 24,
        [10] { 1, 0, 0, 0, 0, 0, 0, 1, 0, 1 },
        [1] { 0 },
        [1] { 2 },
        [1] { 1 },
        [1] { 0.816496580927726 },
        -123, 123, 6, 20.83333333333333, 84.45593854522934, -123, 123,
        [10] { 1, 0, 0, 0, 1, 2, 0, 0, 0, 2 },
        [1] { 1 },
        [1] { 3 },
        [1] { 2 },
        [1] { 0.8164965809277263 },
        -10, 10, 6, 0, 8.18535277187245, -10, 10,
        [10] { 2, 0, 0, 0, 1, 1, 0, 0, 0, 2 },
        [1] { 2 },
        [1] { 2 },
        [1] { 2 },
        [1] { 0 },
        1, 456, 5, 117, 175.8601717274267, 1, 456,
        [10] { 3, 0, 1, 0, 0, 0, 0, 0, 0, 1 },
        [2] { 0, 0 },
        [2] { 3, 2 },
        [2] { 1.333333333333333, 1 },
        [2] { 1.247219128924647, 0.816496580927726 } };;
```

Figure 8: A sample summary file (continued).

```
"Absent Record::__summary__" {
        0, 3, a, a, 0, 0, 0, a, a,
        [10] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        [0] { },
        [0] { },
        [0] { },
        [0] { },
        a, a, 0, 0, 0, a, a,
        [10] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        [1] { 0 },
        [1] { 0 },
        [1] { 0 },
        [1] { 0 },
        0, 0, 0, 0, 0, 0, 0,
        [10] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        [0] { },
        [0] { },
        [0] { },
        [0] { },
        0, 0, 0, 0, 0, 0, 0,
        [10] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        [1] { 0 },
        [1] { 0 },
        [1] { 0 },
        [1] { 0 } };;
```

Figure 9: A sample summary file (continued).