StatDataML:
An XML Format for Statistical Data

October 16, 2003

1 Technical description

1.1 The File Header

The top level '`StatDataML`' element contains one '`description`' and one '`dataset`' element, each optional:

```
<!ELEMENT StatDataML (description?, dataset?)>
<!ATTLIST StatDataML xmlns CDATA
                     #FIXED "http://www.omegahat.org/StatDataML/">
```

Note that all '`StatDataML`' elements per default live in the '`StatDataML`' namespace defined by the URI "`http://www.omegahat.org/StatDataML/`".

1.2 The '`description`' element

The '`description`' element is used to provide meta-information about a dataset—typically not needed for computations on the data itself:

```
<!ELEMENT description (title?, source?, date?, version?,
                       comment?, creator?, properties?)>

<!ELEMENT title (#PCDATA)>
<!ELEMENT source (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT creator (#PCDATA)>
<!ELEMENT properties (list)>
```

It consists of seven elements: '`title`', '`source`', '`date`', '`comment`', '`version`', and '`creator`' are simple strings ('`PCDATA`'), whereas '`properties`' is a '`list`' element (see next section). The '`creator`' element should contain knowledge about the creating application and the *StatDataML* implementation, '`version`' complements '`date`' in uniquely identifying the data set, and '`properties`' offers a well-defined structure to save application-based meta-information.

1.3 The 'dataset' element

We define a '`dataset`' element either as a list or as an array:

```
<!ELEMENT dataset (list | array)>
```

We use arrays and lists as basic "data types" in *StatDataML* because virtually every data object in statistics can be decomposed into a set of "arrays" and "lists" (as in the S language, or the corresponding "arrays" and "cell-arrays" in MATLAB). The basic property of a list is its generic structure (it may contain data of any type), in contrast to arrays whose elements are all of the same type. As a consequence, lists can also represent recursive structures because they can also contain lists.

1.3.1 Lists

A list contains of three elements: '`dimension`', '`properties`', and '`listdata`':

```
<!ELEMENT list (dimension, properties?, listdata)>
<!ELEMENT listdata (list | array | empty)*>
```

The '`dimension`' element may contain several '`dim`' tags, depending on the number of dimensions:

```
<!ELEMENT dimension (dim*)>
<!ELEMENT dim (e*)>
<!ATTLIST dim size CDATA #REQUIRED>
<!ATTLIST dim name CDATA #IMPLIED>
```

Each of them has '`size`' as a required attribute, and may optionally contain up to '`size`' names, specified with '`<e>`'...'`</e>`' tags. In addition, the dimension as a whole can be attributed a name by the optional '`name`' attribute. Note that arrays, like the whole dataset, can also have additional '`properties`' attached, corresponding, e.g., to attributes in S. The '`listdata`' element may either contain arrays (with the actual data), again lists (allowing complex and even recursive structures), or '`empty`' tags (indicating non-existing elements, corresponding to '`NULL`' in S).

1.3.2 Arrays

Arrays are blocks of data objects of the same elementary type with dimension information used for memory allocation and data access (indexing):

```
<!ELEMENT array (dimension, type, properties?, (data | textdata))>
```

The '`dimension`' and '`properties`' elements are identical to the corresponding '`list`' tags. The '`listdata`' block gets replaced by the '`data`' (or '`textdata`') element that contains the data itself. The '`type`' element contains all information about the statistical data type:

```
<!ELEMENT type (logical | categorical | numeric | character | datetime)>

<!ELEMENT logical EMPTY>
<!ELEMENT categorical (label)+>
```

```
<!ELEMENT numeric (integer | real | complex)?>
<!ELEMENT character EMPTY>
<!ELEMENT datetime EMPTY>
```

It must contain exactly one '`logical`', '`categorical`', '`numeric`', '`character`', or '`datetime`' tag. The '`categorical`' tag must—and the '`numeric`' element may—contain additional elements, providing even finer type characterization.

The '`categorical`' tag carries a '`mode`' attribute that can be '`unordered`' ("factors"), '`ordered`', or '`cyclic`' (e.g., days of the week)—'`unordered`' is the default:

```
<!ELEMENT categorical (label)+>
<!ATTLIST categorical mode (unordered | ordered | cyclic) "unordered">
```

In addition, the factor labeling has to be specified by the means of one or more '`label`' tags:

```
<!ELEMENT label (#PCDATA)>
<!ATTLIST label code CDATA #REQUIRED>
```

The '`label`' element has a mandatory '`code`' attribute specifying the levels' integer value, and optionally contains a name. If no name is given, the application should use the numerical code instead. The order of the '`label`' elements also defines the ordering relation of the levels for ordinal data. As an example, consider the type specification of a color factor:

```
<type>
  <categorical mode="unordered">
    <label code="1">red</label>
    <label code="2">green</label>
    <label code="3">blue</label>
    <label code="4">yellow</label>
  </categorical>
</type>
```

In the data section (see below), only the codes will be used.

Finally, the '`numeric`' element may contain a further tag, allowing the distinction of '`integer`', '`real`', and '`complex`' data:

```
<!ELEMENT numeric (integer | real | complex)?>

<!ELEMENT integer (min?, max?)>
<!ELEMENT real (min?, max?)>
<!ELEMENT complex>
```

If '`numeric`' is left empty, the data is assumed to be real. For '`integer`' and '`real`', one optionally can specify the data range using the '`min`' and '`max`' tags, allowing the parsing software both to choose a memory-saving storage mode and to check the data validity:

```
<!ENTITY % RANGE "#PCDATA | posinf | neginf">
<!ELEMENT min (%RANGE;)>
<!ELEMENT max (%RANGE;)>
```

As an example, consider the type specification for the integers from 1 to 10:

```
<type>
  <numeric>
    <integer>
      <min>1</min> <max>10</max>
    <integer/>
  <numeric/>
<type/>
```

The content of '`min`' and '`max`' should be '`<posinf/>`' and '`<neginf/>`' for $+\infty$ and $-\infty$, respectively.

1.3.3 The '`data`' tag

If '`data`' is used (especially recommended for character data), then each element of the array representing an existing value is encapsulated in '`<e>`'...'`</e>`' pairs (or '`<ce>`'...'`</ce>`' for complex numbers). For missing values, '`<na/>`' has to be used, empty values are just represented by '`<e></e>`' (or simply '`<e/>`'):

```
<!ELEMENT data (e|ce|na|T|F)* >

<!ENTITY % REAL "#PCDATA|posinf|neginf|nan">
<!ELEMENT e (%REAL;)* >
<!ELEMENT posinf EMPTY>
<!ELEMENT neginf EMPTY>
<!ELEMENT nan EMPTY>

<!ELEMENT ce (r,i) >
<!ELEMENT r (%REAL;)* >
<!ELEMENT i (%REAL;)* >

<!ELEMENT na EMPTY>

<!ATTLIST e  info CDATA #IMPLIED>
<!ATTLIST ce info CDATA #IMPLIED>
<!ATTLIST na info CDATA #IMPLIED>
<!ATTLIST T info CDATA #IMPLIED>
<!ATTLIST F info CDATA #IMPLIED>
```

As an example, consider a character dataset formed by color names, with one value missing (after '`green`'), and one being empty (after '`blue`'). The corresponding '`data`' section would appear as follows:

```
<data>
  <e>red</e> <e>green</e> <na/> <e>blue</e> <e></e> <e>yellow</e>
</data>
```

If the colors were coded as factor levels, the example would become:

```
<data>
  <e>1</e> <e>2</e> <na/> <e>3</e> <e></e> <e>4</e>
</data>
```

(Note that the association between numbers and labels is defined in the '`type`' section as mentioned above.)
'`<e>`', and '`<ce>`', and '`<na/>`' tags (and also '`<T/>`' and '`<F/>`', see above) can carry an optional '`info`' attribute, allowing the storage of meta-information:

```
<e>120</e> <e info="unsure">123<e/> <na info="data deleted"/>
```

IEEE Number Format

Computer systems represent numbers in different ways, depending on their hardware architecture. We require the number format to follow the IEEE Standard for Binary Floating Point Arithmetic (Institute of Electrical and Electronics Engineers, 1985), implemented by most programming languages and system libraries. However, the IEEE special values '`+Inf`', '`-Inf`' and '`NaN`' must explicitly be specified by '`<posinf/>`', '`<neginf/>`', and '`<nan/>`', respectively, to facilitate the parsing process in case the IEEE standard was not implemented (we are not distinguishing between the 'Quiet' and 'Signalling' variants of `NaN` provided by the IEEE standard, the 'signalling' one—if ever used—being more close to `NA` values for which we define a special symbol). These special values could appear, e.g., as follows:

```
<data>
  <e>1.23</e> <e><posinf/></e> <e><nan/></e> <e>2.43</e>
</data>
```

When an application reads a *StatDataML* file, the implementation is responsible for the correct casts, i.e. for choosing the appropriate number representation in the computer system.

Complex Numbers

Complex numbers are enclosed in '`<ce>`'...'`</ce>`' tags, containing exactly one '`<r>`'...'`</r>`' tag (for the real part) and one '`<i>`'...'`</i>`' tag (for the imaginary part). Apart from that, the same rules as for '`<e>`'...'`</e>`' apply:

```
<data>
  <ce> <r>12.4</r> <i>1</i> </ce>
</data>
```

Logical Values

The 'true' and 'false' values are represented by the special tags '`<T/>`' and '`<F/>`':

```
<data>
  <T/> <F/>
</data>
```

Date and Time Information

Data of type '`datetime`' has to follow the ISO 8601 specification (see International Organization for Standardization, 2000). *StatDataML* should only make use of the complete representation in extended format of the combined calendar date and time of the day representation:

CCYY-MM-DDThh:mm:ss±hh:mm

where the characters represent Century (C), Year (Y), Month (M), Day (D), Time designator (T; indicates the start of time elements), Hour (h), Minutes (m) and Seconds (s). For example, the 12th of March 2001 at 12 hours and 53 minutes, UTC+1, would be represented as: `2001-03-12T12:53:00+01:00`.

1.3.4 The 'textdata' tag

For memory and storage space efficiency, we also define '`textdata`', a second way of writing data blocks using arbitrary characters (typically whitespace) for separating elements instead of '`<e>`'...'`</e>`':

```
<!ELEMENT textdata (#PCDATA) >
<!ATTLIST textdata sep           CDATA " &#x000A;&#x000D;"
                   na.string     CDATA "NA"
                   null.string   CDATA "NULL"
                   posinf.string CDATA "+Inf"
                   neginf.string CDATA "-Inf"
                   nan.string    CDATA "NaN"
                   true          CDATA "1"
                   false         CDATA "0">
```

In this case, the complete data block is included in a single XML tag: because only a single character is used as separator, one needs 6 bytes less per element. The use of '`textdata`' even provides more compact results when compression tools (such as *zip*) are used, and is recommended if such tools are not available or if their use is not desirable. The set of separator characters is defined by the optional attribute '`sep`' (the default set consists of three characters: space, line feed, and carriage return). The attributes '`na.string`' and '`null.string`' define the strings to be interpreted as missing or empty values (default: '`NA`' and '`NULL`'). '`posinf.string`', '`neginf.string`', and '`nan.string`' are used to specify the corresponding IEEE special values. The '`true`' and '`false`' attributes can be used to change the default representation of logical values ('`1`' and '`0`').

An additional "advantage" is that textdata blocks are not parsed by the XML parser, which can drastically reduce the memory footprint when reading a file, because many parsers represent the complete XML data as a nested tree. This results in one branch for each array element and typically needs much more memory than just the element itself. Our color-example could look similar to the following:

```
<textdata na.string="N/A" null.string="EMPTY">
  red green EMPTY blue N/A yellow
</textdata>
```

In this example, we use the default separator set ("

"), but alternatively, we could have defined a different set of symbols with the 'sep'-attribute, such as colons or semi-colons.

1.4 Implementation issues

Interfaces implementing *StatDataML* should provide options for setting conversion strings for the 'NA', $\pm\infty$ and 'NaN' entities if they are not supported, but with no defaults. Unsupported elements with no default conversion should cause an error, thus forcing the user to explicitly specify a conversion rule. All conversions effectively done should be reported by a warning message.

2 Examples

2.1 Demo Sessions

2.1.1 Exchanging data between R and MATLAB

This example shows how to bring the 'iris' data set to MATLAB and the 'durer' data to R.

We start in R ...

```
R : Copyright 2003, The R Development Core Team
Version 1.6.2  (2003-01-10)

## load the StatDataML package
> library(StatDataML)
Loading required package: XML

## load the iris data
> data(iris)

## show the first observation
> iris[1,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa

## write StatDataML file
> writeSDML(iris, file = "iris.sdml")
```

. . . continue in MATLAB **. . .**

```
                          < M A T L A B >
                Copyright 1984-2000 The MathWorks, Inc.
                     Version 6.0.0.88 Release 12

>> path(path, 'StatDataML')
>> x = readsdml('iris.sdml', 'list.as.structarray')

>> x(1)

ans =

    Sepal.Length: 5.1000
     Sepal.Width: 3.5000
    Petal.Length: 1.4000
     Petal.Width: 0.2000
         Species: {'setosa'}

>> load durer
>> whos X, c = cellstr(caption)

  Name      Size          Bytes  Class
  X       648x509       2638656  double array

c =

'Albrecht Durer's Melancolia.'
'Can you find the matrix?'

>> writesdml(X, 'durer.sdml', 'TITLE', c{1}, 'TEXTDATA', 'TRUE')
```

. . . and return to R **again:**

```
> durer <- readSDML("durer.sdml", read.description = TRUE)
> attr(durer, "SDMLdescription")$title

[1] "Albrecht Durer's Melancolia."
```

2.1.2 An example with more complex data

We show how a sample task (creating a simple, matrix-like structure, writing a *StatDataML* file in R, reading this file in MATLAB) is realized.

A session started with R ...

```
R : Copyright 2002, The R Development Core Team
Version 1.6.1 (2002-11-01)

## load the StatDataML package
> library(StatDataML)
Loading required package: XML

## Create a simple data structure
> X <- list(matrix(c(1, 2, 3, 4), 2, 2))
> X[[2]] <- 12 + 3i
> X[[3]] <- "Test"
> X[[4]] <- list(a = "Test 2", b = 33.44)
> dim(X) <- c(2, 2)

## Show what we got
> X
     [,1]        [,2]
[1,] "Numeric,4" "Character,1"
[2,] "Complex,1" "List,2"

> X[[1,1]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> X[[1,2]]
[1] "Test"

> X[[2,1]]
[1] 12+3i

> X[[2,2]]
$a
[1] "Test 2"
$b
[1] 33.44

## write it to the .sdml file
> writeSDML(X, "test.sdml")
```

. . . and finished in MATLAB**:**

```
                        < M A T L A B >
            Copyright 1984-2000 The MathWorks, Inc.
                 Version 6.0.0.88 Release 12

>> path(path, 'StatDataML')
>> X = readsdml('test.sdml')

X =

          [2x2 double]    'Test'
    [12.0000+ 3.0000i]    [1x1 struct]

>> X{1}

ans =

     1     3
     2     4

>> X{2}

ans =

  12.0000 + 3.0000i

>> X{3}

ans =

Test

>> X{4}

ans =

    a: 'Test 2'
    b: 33.4400
```

2.2 The integers from 1 to 10

```
<?xml version="1.0"?>
<!DOCTYPE StatDataML PUBLIC "StatDataML.dtd" "StatDataML.dtd">

<StatDataML xmlns="http://www.omegahat.org/StatDataML/">

<description>
  <title>The integers from 1 to 10</title>
  <source>MATLAB</source>
  <date>2001-10-10T14:40:01+0200</date>
  <version></version>
  <comment></comment>
  <creator>MATLAB-6.0.0.88 (R12):StatDataML_1.0-0</creator>
</description>

<dataset>
  <array>
    <dimension>
      <dim size="10"></dim>
    </dimension>
    <type> <numeric> <integer/> <numeric/> <type/>
    <data>
      <e>1</e> <e>2</e> <e>3</e> <e>4</e> <e>5</e>
      <e>6</e> <e>7</e> <e>8</e> <e>9</e> <e>10</e>
    </data>
  </array>
</dataset>

</StatDataML>
```

2.3 A more complex example

The following example represents a table with two variables (which could be, e.g., a dataframe in S, and a structure in MATLAB): one character variable 'a' (with one missing value), and one numerical variable 'b'.

a	A	B		D	E
b	0.5	$+\infty$	4.5	NaN	1.0

```
<?xml version="1.0"?>
<!DOCTYPE StatDataML PUBLIC "StatDataML.dtd" "StatDataML.dtd">

<StatDataML xmlns="http://www.omegahat.org/StatDataML/">

<description>
  <title>A small dataframe</title>
  <source>MATLAB</source>
  <date>2001-10-10T14:43:02+0200</date>
  <version></version>
  <comment></comment>
  <creator>MATLAB-6.0.0.88 (R12):StatDataML_1.0-0</creator>
</description>

<dataset>
  <list>
    <dimension>
      <dim size="2"> <e>a</e> <e>b</e> </dim>
    </dimension>

    <listdata>
      <array>
        <dimension>
          <dim size="5"/>
        </dimension>
        <type> <character/> <type/>
        <data>
          <e>A</e> <e>B</e> <na/> <e>D</e> <e>E</e>
        </data>
      </array>
      <array>
        <dimension>
          <dim size="5"/>
        </dimension>
        <type> <numeric/> <type>
        <data>
          <e>0.5</e> <e><posinf/></e> <e>4.5</e> <e><nan/></e> <e>1.0</e>
        </data>
      </array>
    </listdata>
  </list>
</dataset>

</StatDataML>
```

3 Implementation

Currently we have support for R, MATLAB and Octave, and converters for SPSS and Gnumeric are under development. The complete package including all implementations can be found at the homepage of the "Omegahat" project, which aims at providing a variety of open-source software for statistical applications (focusing on web-based software, Java, the Java virtual machine, and distributed computing): http://www.omegahat.org/StatDataML/.

The software for the R system alone is also provided as an R package from the software archive of the R project for statistical computing (http://cran.r-project.org/src/contrib/StatDataML_1.0-4.tar.gz). It provides an implementation of *StatDataML* I/O routines for R. The two functions '`writeSDML`' and '`readSDML`' implement writing and reading for *StatDataML* files. With this implementation, it is possible to write and read R data objects without loss of information.

All implementations make use of XML functionality provided by `libxml`, the XML C library for gnome (http://www.xmlsoft.org/). The R implementation of *StatDataML* in addition requires Duncan Temple Lang's XML package, providing general XML parsing for S engines (http://cran.r-project.org/src/contrib/Omegahat/XML.tar.gz).

References

Institute of Electrical and Electronics Engineers (1985). *IEEE Standard 754-1985 (R 1990), Standard for Binary Floating-Point Arithmetic.*

International Organization for Standardization (2000). *ISO 8601:2000, Data elements and interchange formats - Information Interchange - Representation of dates and times.*

D. Meyer, F. Leisch, T. Hothorn, and K. Hornik. StatDataML: An XML format for statistical data. In W. Härdle and B. Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 545–550. Physika Verlag, Heidelberg, Germany, 2002. ISBN 3-7908-1517-9.

D. Meyer, F. Leisch, T. Hothorn, and K. Hornik. StatDataML: An XML format for statistical data. *Computational Statistics*. Forthcoming.

Appendix: the *StatDataML* `.dtd` file

```
<!-- StatDataML DTD version="1.0" -->

<!ELEMENT StatDataML (description?, dataset?)>
<!ATTLIST StatDataML xmlns CDATA #FIXED "http://www.omegahat.org/StatDataML/">

<!-- document description tags -->

<!ELEMENT description (title?, source?, date?, version?,
                      comment?, creator?, properties?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT source (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT creator (#PCDATA)>
<!ELEMENT properties (list)>

<!-- basic elements -->

<!ELEMENT dataset (list | array)>

<!ELEMENT list (dimension, properties?, listdata)>
<!ELEMENT listdata (list | array | empty)*>
<!ELEMENT empty EMPTY>

<!ELEMENT array (dimension, type, properties?, (data | textdata))>

<!-- dimension elements -->

<!ELEMENT dimension (dim*)>
<!ELEMENT dim (e*)>
<!ATTLIST dim size CDATA #REQUIRED>
<!ATTLIST dim name CDATA #IMPLIED>

<!-- type elements -->

<!ELEMENT type (logical | categorical | numeric | character | datetime)>

<!ELEMENT logical EMPTY>

<!ELEMENT categorical (label)+>
<!ATTLIST categorical mode (unordered | ordered | cyclic) "unordered">
<!ELEMENT label (#PCDATA)>
<!ATTLIST label code CDATA #REQUIRED>

<!ELEMENT numeric (integer | real | complex)?>
<!ELEMENT integer (min?, max?)>
<!ELEMENT real (min?, max?)>
<!ELEMENT complex EMPTY>
<!ENTITY % RANGE "#PCDATA | posinf | neginf">
<!ELEMENT min (%RANGE;)*>
<!ELEMENT max (%RANGE;)*>
```

```
<!ELEMENT character EMPTY>

<!ELEMENT datetime EMPTY>

<!-- data/textdata tags -->

<!ELEMENT data (e|ce|na|T|F)* >

<!ELEMENT textdata (#PCDATA) >
<!ATTLIST textdata sep CDATA " &#x000A;&#x000D;"
                   na.string CDATA "NA"
                   null.string CDATA "NULL"
                   posinf.string CDATA "+Inf"
                   neginf.string CDATA "-Inf"
                   nan.string    CDATA "NaN"
                   true CDATA "1"
                   false CDATA "0">

<!-- e/ce/na elements -->

<!ELEMENT na EMPTY>

<!ENTITY % REAL "#PCDATA | posinf | neginf | nan">

<!ELEMENT e (%REAL;)*>
<!ELEMENT posinf EMPTY>
<!ELEMENT neginf EMPTY>
<!ELEMENT nan EMPTY>

<!ELEMENT ce (r,i) >
<!ELEMENT r (%REAL;)*>
<!ELEMENT i (%REAL;)*>

<!ELEMENT T EMPTY>
<!ELEMENT F EMPTY>

<!ATTLIST e  info CDATA #IMPLIED>
<!ATTLIST ce info CDATA #IMPLIED>
<!ATTLIST na info CDATA #IMPLIED>
<!ATTLIST T  info CDATA #IMPLIED>
<!ATTLIST F  info CDATA #IMPLIED>
```