ExpressionView file format

Gábor Csárdi

April 16, 2015

Contents

1 Introduction

ExpressionView is an interactive visualization tool for biclusters in gene expression data. The software has two parts. The first part is an ordering algorithm, written in GNU R, that reorders the rows and columns of the gene expression matrix to make (potentially overlapping) biclusters more visible. The second part is the interactive tool, written in Adobe Flex. It runs in an Adobe Flash enabled web browser. The user can export the ordered gene expression matrix, with additional meta-data from R to a data file, that can be openened by the Adobe Flash application. In this document we briefly discuss the format of this data file.

2 The file format

The EVF data file, used by ExpressionView, is a standard XML file. The R package contains an XML Schema file that describes the exact format. In the following we will show this schema file and explain its parts step by step, while also showing samples from an example EVF data file. Parts of the schema file appear in green boxes, EVF file code snipplets are in blue boxes.

2.1 Header and main parts

The schema file starts with a standard header:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:annotation>
    <xsd:documentation xml:lang="en">
     ExpressionView file format schema, version 1.1.
     ExpressionView is a tool to visualize modules (biclusters) in
     gene expression data. Please see http://www.unil.ch/cbg for
     details. Copyright 2010 UNIL DGM Computational Biology Group
    </xsd:documentation>
  </xsd:annotation>
```

This is the 1.1 version of the EVF file. ExpressionView can also read the older 1.0 version.

```
<xsd:element name="evf" type="evfType" />
```

An EVF file contains a single `<evf>` tag. It has the following parts:

```
<xsd:complexType name="evfType">
  <xsd:sequence>
    <xsd:element name="summary" type="SummaryType" />
    <xsd:element name="experimentdata" type="ExperimentDataType" />
    <xsd:element name="genes" type="GenesType" />
    <xsd:element name="samples" type="SamplesType" />
    <xsd:element name="modules" type="ModulesType" />
    <xsd:element name="data" type="xsd:base64Binary" />
  </xsd:sequence>
</xsd:complexType>
```

The `<summary>` tag contains information about the data set, such as the number of genes, samples, modules, etc. `<experimentdata>` is for experiment meta-data, i.e. the lab where it was performed, the abstract of the related publication and possibly more. `<genes>` and `<samples>` have the gene and sample meta data. `<modules>` defines the biclusters. Finally `<data>` contains the expression values themselves. Let us see all of these tags in detail now.

2.2 Summary

This is the type of the `<summary>` tag:

```
<xsd:complexType name="SummaryType">
  <xsd:sequence>
    <xsd:element name="description" minOccurs="0" type="xsd:string" />
    <xsd:element name="version" type="xsd:string" />
    <xsd:element name="dataorigin" minOccurs="0" type="xsd:string" />
    <xsd:element name="xaxislabel" minOccurs="0" type="xsd:string" />
    <xsd:element name="yaxislabel" minOccurs="0" type="xsd:string" />
    <xsd:element name="nmodules" type="xsd:nonNegativeInteger" />
    <xsd:element name="ngenes" type="xsd:positiveInteger" />
    <xsd:element name="nsamples" type="xsd:positiveInteger" />
  </xsd:sequence>
</xsd:complexType>
```

The `<summary>` tag optionally contains the description of the data file (`<description>` tag), this is displayed above the main gene expression window in Expression-View. The `<version>` tag must be `1.1` for files for the format we are discussing here. The `<dataorigin>` tag is optional and has the value 'eisa' for modules generated with the ISA algorithm [1] and exported using the ExpressionView R package. `<xaxislabel>` and `<yaxislabel>` are optional axis labels for the gene expression plot. The last three tags (`<nmodules>`, `<ngenes>` and `<nsamples>`) are required and give the number of modules, number of genes and number of samples in the data set.

A sample EVF file header, together with the `<summary>` tag looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<evf>
  <summary>
    <description>ExpressionView data file</description>
    <version>1.1</version>
    <dataorigin>eisa</dataorigin>
    <nmodules>8</nmodules>
    <ngenes>3522</ngenes>
    <nsamples>128</nsamples>
  </summary>
```

2.3 Experiment meta-data

The `<experimentdata>` tag is next, this contains the experiment meta-data. It has the following parts:

```
<xsd:complexType name="ExperimentDataType">
  <xsd:all>
    <xsd:element name="title" minOccurs="0" type="xsd:string" />
    <xsd:element name="name" minOccurs="0" type="xsd:string" />
    <xsd:element name="lab" minOccurs="0" type="xsd:string" />
```

3

```
    <xsd:element name="abstract" minOccurs="0" type="xsd:string" />
    <xsd:element name="url" minOccurs="0" type="xsd:anyURI" />
    <xsd:element name="annotation" minOccurs="0" type="xsd:string" />
    <xsd:element name="organism" minOccurs="0" type="xsd:string" />
  </xsd:all>
</xsd:complexType>
```

All fields are optional here: `<title>`, `<name>`, `<lab>`, `<abstract>`, `<url>`, `<annotation>`, `<organism>`. They are collected and shown in the *Experiment* tab in the interactive ExpressionView viewer. The `<annotation>` tag should contain the name of the chip on which the experiment was performed. If the `<organism>` tag contains 'Homo sapiens', then ExpressionView links genes with the Gene Cards homepage, for other organism is uses Entrez Gene. The tags within `experimentdata` can be specified in arbitrary order.

Here is an example for the `<experimentdata>` tag:

```
<experimentdata>
  <title>Gene expression profile of adult T-cell acute
    lymphocytic leukemia identifies distinct subsets of
    patients with different response to therapy and
    survival.
  </title>
  <name>Chiaretti et al.</name>
  <lab>Department of Medical Oncology, ...</lab>
  <abstract>Gene expression profiles were examined ...</abstact>
  <url>http://...</url>
  <annotation>hgu95av2</annotation>
  <organism>Homo sapiens</organism>
</experimentdata>
```

2.4   Genes

The next tag within `<evf>` is `<genes>`, its type os defined as

```
<xsd:complexType name="GenesType">
  <xsd:sequence>
    <xsd:element name="genetags" type="GeneTagsType" />
    <xsd:element name="gene" maxOccurs="unbounded" type="GeneType" />
  </xsd:sequence>
</xsd:complexType>
```

It has two main parts, a `<genetags>` tag first, and then a number of `<gene>` tags, one for each gene in the data set.

Before we continue with the gene tags, we define a new tag type, that is required to contain a `name` attribute. We call this type `ExtraType`. This tag type can be used to add any kind of meta data to the table in the *Genes* tab in ExpressionView.

```
<xsd:complexType name="ExtraType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="name" type="xsd:string" use="required" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

The `<genetags>` tag has one required and many optional subtags:

```
<xsd:complexType name="GeneTagsType">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:string" />
    <xsd:element name="name" minOccurs="0" type="xsd:string" />
    <xsd:element name="symbol" minOccurs="0" type="xsd:string" />
    <xsd:element name="entrezid" minOccurs="0" type="xsd:string" />
    <xsd:element name="x" minOccurs="0" maxOccurs="unbounded"
               type="ExtraType" />
  </xsd:sequence>
</xsd:complexType>
```

The `<id>` must be an integer number, that starts with one and goes up to the number of genes. It is used as an id that is refered from other tags in the document. `<name>` is typically the name of the probe-set, but it can be used for other purposes as well, `<symbol>` is typically the canonical gene symbol. `<entrezid>` is the Entrez gene id. If a given probeset does not map to a known gene, then `<symbol>` and `<entrezid>` can be set to `NA`. Finally, the gene tags might contain any number of `<x>` tags, each defining an additional column in the *Genes* tab in ExpressionView. These tags must have a `name` attribute, which is refered by the tags of the individual genes. Here is an example gene tags section:

```
<genetags>
  <id>#</id>
  <name>Name</name>
  <symbol>Symbol</symbol>
  <entrezid>EntrezID</entrezid>
  <x name="chr">Chromosome</x>
</genetags>
```

We define an extra column for the gene table, this will give the chromosome on which the gene is located.

After the gene tags, we have a typically large number of genes in the file. The definition of the `<gene>` tag:

```
<xsd:complexType name="GeneType">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:string" />
    <xsd:element name="name" minOccurs="0" type="xsd:string" />
    <xsd:element name="symbol" minOccurs="0" type="xsd:string" />
    <xsd:element name="entrezid" minOccurs="0" type="xsd:string" />
    <xsd:element name="x" minOccurs="0" maxOccurs="unbounded"
```

```
            type="ExtraType"/>
  </xsd:sequence>
</xsd:complexType>
```

The `<gene>` tags must have exactly the same subtags as the ones given in the gene tags section. For example, assuming the `<genetags>` example above we can have:

```
<gene>
  <id>1</id>
  <name>33500_i_at</name>
  <symbol>NA</symbol>
  <entrezid>NA</entrezid>
  <x name="chr">NA</x>
</gene>
<gene>
  <id>2</id>
  <name>40990_at</name>
  <symbol>TSPAN5</symbol>
  <entrezid>10098</entrezid>
  <x name="chr">4</x>
</gene>
...
```

Observe, how the extra tag for the chromosomes is referenced here.

2.5   Samples

The samples and their associated data have a format similar to the genes:

```
<xsd:complexType name="SamplesType">
  <xsd:sequence>
    <xsd:element name="sampletags" type="SampleTagsType" />
    <xsd:element name="sample" maxOccurs="unbounded"
                 type="SampleType" />
  </xsd:sequence>
</xsd:complexType>
```

The `<samples>` tag has two parts, the first defines the sample meta data entries and the second part contains one `<sample>` tag for each sample in the data set.

```
<xsd:complexType name="SampleTagsType">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:string" />
    <xsd:element name="name" minOccurs="0" type="xsd:string" />
    <xsd:element name="x" minOccurs="0" maxOccurs="unbounded"
                 type="ExtraType" />
  </xsd:sequence>
</xsd:complexType>
```

The <id> and <name> sample tags are required. <id> contains numeric ids, from one, up to the number of samples in the data set. These are referenced by other tags, e.g. the ones that define the modules. <name> is an arbitrary string, it is typically the sample name in the R ExpressionSet object, that was used to find the biclusters. Similarly to the genes, any number of additional <x> tags can be added, each with a unique name attribute, to define additional information about the samples.

Here is an example <sampletags> tag:

```
<sampletags>
  <id>#</id>
  <name>Name</name>
  <x name="diagnosis"> Date of diagnosis</x>
  <x name="sex"> Gender of the patient</x>
  <x name="age"> Age of the patient at entry</x>
  <x name="BT"> does the patient have B-cell or T-cell ALL</x>
</sampletags>
```

We defined four extra tags for the samples in this example.

Then the samples follow:

```
<xsd:complexType name="SampleType">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:string" />
    <xsd:element name="name" minOccurs="0" type="xsd:string" />
    <xsd:element name="x" minOccurs="0" maxOccurs="unbounded"
                 type="ExtraType" />
  </xsd:sequence>
</xsd:complexType>
```

Here is an example, that corresponds to the <sampletags> entry given above:

```
<sample>
  <id>1</id>
  <name>01010</name>
  <x name="diagnosis">3/29/2000</x>
  <x name="sex">2</x>
  <x name="age">19</x>
  <x name="BT">3</x>
</sample>
<sample>
  <id>2</id>
  <name>04010</name>
  <x name="diagnosis">10/30/1997</x>
  <x name="sex">1</x>
  <x name="age">18</x>
  <x name="BT">2</x>
</sample>
...
```

2.6 Modules

Again, the `<modules>` tag has a syntax that is similar to the syntax of the genes and the samples:

```
<xsd:complexType name="ModulesType">
  <xsd:sequence>
    <xsd:element name="moduletags" type="ModuleTagsType" />
    <xsd:element name="gotags" minOccurs="0" type="GoTagsType" />
    <xsd:element name="keggtags" minOccurs="0" type="KeggTagsType" />
    <xsd:element name="module" maxOccurs="unbounded"
                 type="ModuleType" />
  </xsd:sequence>
</xsd:complexType>
```

There are two required and two optional parts. The first part is `<module-tags>`, this defines the meta-data associated with the modules. The `<gotags>` and `<keggtags>` tags are optional, they are only present if the file contains Gene Ontology and/or KEGG pathway enrichment calculation results for the biclusters. Finally, there is a list of `<module>` tags, one for each bicluster.

The type of `<moduletags>`:

```
<xsd:complexType name="ModuleTagsType">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:string" />
    <xsd:element name="name" minOccurs="0" type="xsd:string" />
    <xsd:element name="iterations" minOccurs="0" type="xsd:string" />
    <xsd:element name="oscillation" minOccurs="0" type="xsd:string" />
    <xsd:element name="thr_row" minOccurs="0" type="xsd:string" />
    <xsd:element name="thr_col" minOccurs="0" type="xsd:string" />
    <xsd:element name="freq" minOccurs="0" type="xsd:string" />
    <xsd:element name="rob" minOccurs="0" type="xsd:string" />
    <xsd:element name="rob_limit" minOccurs="0" type="xsd:string" />
    <xsd:element name="x" minOccurs="0" maxOccurs="unbounded"
                 type="ExtraType" />
  </xsd:sequence>
</xsd:complexType>
```

There is only one tag required in `<moduletags>`, the `<id>`, which is a numeric id, going from 1 to the number of modules. The rest is optional, and it is also possible to create any kind of extra tags with the `<x>` tag. `<name>` is an arbitrary character string. The other optional tags are mainly defined to modules coming from the ISA algorithm: `<iterations>` gives the number of ISA iterations needed to find the module, `<oscillation>` specifies the oscillation cycle length for oscillating modules. It is mostly zero, meaning that the module does not oscillate. `<thr_row>` is the ISA gene threshold, `<thr_col>` is the ISA condition threshold. `<freq>` is the number of ISA seeds that converged to the module, `<rob>` is its robustness score. `<rob_limit>` is the robustness threshold that was used to filter the module.

Here is an example `<moduletags>` tag:

```
<moduletags>
  <id>#</id>
  <name>Name</name>
  <iterations>Iterations</iterations>
  <oscillation>Oscillation cycle</oscillation>
  <thr_row>Gene threshold</thr_row>
  <thr_col>Column threshold</thr_col>
  <freq>Frequency</freq>
  <rob>Robustness</rob>
  <rob_limit>Robustness limit</rob_limit>
</moduletags>
```

The optional `<gotags>` and `<keggtags>` tags are discussed in Section 2.7, let us now see how the data for a single module looks like:

```
<xsd:complexType name="ModuleType">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:string" />
    <xsd:element name="name" minOccurs="0" type="xsd:string" />
    <xsd:element name="iterations" minOccurs="0" type="xsd:string" />
    <xsd:element name="oscillation" minOccurs="0" type="xsd:string" />
    <xsd:element name="thr_row" minOccurs="0" type="xsd:string" />
    <xsd:element name="thr_col" minOccurs="0" type="xsd:string" />
    <xsd:element name="freq" minOccurs="0" type="xsd:string" />
    <xsd:element name="rob" minOccurs="0" type="xsd:string" />
    <xsd:element name="rob_limit" minOccurs="0" type="xsd:string" />
    <xsd:element name="x" minOccurs="0" maxOccurs="unbounded"
                 type="ExtraType" />
    <xsd:element name="containedgenes" type="integerList" />
    <xsd:element name="genescores" type="doubleList" />
    <xsd:element name="containedsamples" type="integerList" />
    <xsd:element name="samplescores" type="doubleList" />
    <xsd:element name="intersectingmodules" type="integerList" />
    <xsd:element name="gos" minOccurs="0" type="GosType" />
    <xsd:element name="keggs" minOccurs="0" type="KeggsType" />
  </xsd:sequence>
</xsd:complexType>
```

The first couple of fields refer to the ones defined in the `<moduletags>` section above. The others are: `<containedgenes>`, a space separated list of gene ids, for the genes that are included in the module; `<genescores>`, the gene scores of these genes, in the order of the list in `<containedgenes>`; `<containedsamples>`, the ids of the samples in the module; `<samplescores>`, the scores for these samples; `<intersectingmodules>`, the ids of the modules that have an overlap with the given module. `<gos>` and `<keggs>` are optional and describe the Gene Ontology and KEGG pathway enrichment of the given module, see their details in Section 2.7.

Here is an example `<module>` tag:

```
<module>
  <id>1</id>
```

```
  <name>module 1</name>
  <iterations>22</iterations>
  <oscillation>0</oscillation>
  <thr_row>2.7</thr_row>
  <thr_col>1.4</thr_col>
  <freq>1</freq>
  <rob>21.98</rob>
  <rob_limit>21.98</rob_limit>
  <containedgenes>214 215 216 217 218 219 220 221 222
  </containedgenes>
  <genescores>-0.94 -0.88 0.74 -0.76 -1.00 -0.84 -0.74 -0.76 -0.85
  </genescores>
  <containedsamples>63 64 65 54 66</containedsamples>
  <samplescores>-0.62 -1.00 -0.77 -0.40 -0.28</samplescores>
  <intersectingmodules>7</intersectingmodules>
  <gos>
    ... see below ...
  </gos>
  <keggs>
    ... see below ...
  </keggs>
</module>
```

We refered to the integer list and double list data types above, now we define them, this is the integer list:

```
<xsd:simpleType name="integerList">
  <xsd:list itemType="xsd:positiveInteger"/>
</xsd:simpleType>
```

and similarly for the double list:

```
<xsd:simpleType name="doubleList">
  <xsd:list itemType="xsd:double"/>
</xsd:simpleType>
```

2.7   Gene Ontology and KEGG pathway enrichment

EVF files can optionally contain Gene Ontology (GO) and/or KEGG pathway enrichment calculation results. In this case `<moduletags>` is followed by a `<gotags>` and/or a `<keggtags>` tag. The definition of `<gotags>`:

```
<xsd:complexType name="GoTagsType">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:string" />
    <xsd:element name="go" type="xsd:string" />
    <xsd:element name="term" type="xsd:string" />
    <xsd:element name="ontology" type="xsd:string" />
    <xsd:element name="pvalue" type="xsd:string" />
    <xsd:element name="oddsratio" type="xsd:string" />
    <xsd:element name="expcount" type="xsd:string" />
    <xsd:element name="count" type="xsd:string" />
    <xsd:element name="size" type="xsd:string" />
```

```
    </xsd:sequence>
</xsd:complexType>
```

`<id>` is a numeric id, starting from one, up to the total number of enriched GO categories; `<go>` is the GO id; `<term>` is the GO term; `<ontology>` is the GO ontology of the term, its possible values are: `BP`, `CC`, `MF`, standing for *Biological process*, *Cellular component* and *Molecular function*; `<pvalue>` is the enrichment *p*-value; `<oddsratio>` is the odds ratio; `<expcount>` is the expected number of genes in the intersection, by chance; `<count>` is the number of genes in the intersection; `<size>` is the size of the GO term, i.e. the number of genes (in the current gene universe) that are annotated with the enriched term.

```
<xsd:complexType name="KeggTagsType">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:string" />
    <xsd:element name="kegg" type="xsd:string" />
    <xsd:element name="pathname" type="xsd:string" />
    <xsd:element name="pvalue" type="xsd:string" />
    <xsd:element name="oddsratio" type="xsd:string" />
    <xsd:element name="expcount" type="xsd:string" />
    <xsd:element name="count" type="xsd:string" />
    <xsd:element name="size" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
```

The tags for `<keggtags>` is almost the same as for `<gotags>`, but here `<kegg>` id the KEGG pathway ID, and `<pathname>` is the name of the pathway. The rest is the same.

Part of an EVF file with the `<gotags>` and `<keggtags>` tags:

```
<gotags>
  <id>#</id>
  <go>GO</go>
  <term>Term</term>
  <ontology>Ontology</ontology>
  <pvalue>PValue</pvalue>
  <oddsratio>OddsRatio</oddsratio>
  <expcount>ExpCount</expcount>
  <count>Count</count>
  <size>Size</size>
</gotags>
<keggtags>
  <id>#</id>
  <kegg>KEGG</kegg>
  <pathname>Path Name</pathname>
  <pvalue>PValue</pvalue>
  <oddsratio>OddsRatio</oddsratio>
  <expcount>ExpCount</expcount>
  <count>Count</count>
  <size>Size</size>
</keggtags>
```

The actual enrichment data is given in the `<module>` tags, these can contain a `<gos>` and/or a `<keggs>` tag with the enrichment p-values and other statistics.

```
<xsd:complexType name="GosType">
  <xsd:sequence>
    <xsd:element name="go" minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```

`<gos>` is a list of `<go>` tags.

```
<xsd:complexType name="KeggsType">
  <xsd:sequence>
    <xsd:element name="kegg" minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```

`<keggs>` is a list of `<kegg>` tags.

The subtags within a `<go>` or a `<kegg>` tag refer to the tags already listed above in the `<gotags>` and `<keggtags>` tags:

```
<xsd:complexType name="go">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:positiveInteger" />
    <xsd:element name="go" type="xsd:string" />
    <xsd:element name="ontology" type="xsd:string" />
    <xsd:element name="pvalue" type="xsd:double" />
    <xsd:element name="oddsratio" type="xsd:double" />
    <xsd:element name="expcount" type="xsd:double" />
    <xsd:element name="count" type="xsd:nonNegativeInteger" />
    <xsd:element name="size" type="xsd:nonNegativeInteger" />
  </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="kegg">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:positiveInteger" />
    <xsd:element name="kegg" type="xsd:string" />
    <xsd:element name="pathname" type="xsd:string" />
    <xsd:element name="pvalue" type="xsd:double" />
    <xsd:element name="oddsratio" type="xsd:double" />
    <xsd:element name="expcount" type="xsd:double" />
    <xsd:element name="count" type="xsd:nonNegativeInteger" />
    <xsd:element name="size" type="xsd:nonNegativeInteger" />
  </xsd:sequence>
</xsd:complexType>
```

Here is an example, this is part of a `<module>` tag:

```
<gos>
  <go>
    <id>1</id>
    <go>GO:0002376</go>
    <term>immune system process</term>
    <ontology>BP</ontology>
```

```
        <pvalue>1.64e-04</pvalue>
        <oddsratio>7.03</oddsratio>
        <expcount>3.95</expcount>
        <count>16</count>
        <size>353</size>
    </go>
    <go>
        <id>2</id>
        <go>GO:0002504</go>
        <term>antigen processing and presentation of peptide or
          polysaccharide antigen via MHC class II</term>
        <ontology>BP</ontology>
        <pvalue>1.06e-03</pvalue>
        <oddsratio>79.95</oddsratio>
        <expcount>0.10</expcount>
        <count>4</count>
        <size>9</size>
    </go>
</gos>
<keggs>
    <kegg>
        <id>1</id>
        <kegg>05310</kegg>
        <pathname>Asthma</pathname>
        <pvalue>0.02</pvalue>
        <oddsratio>31.60</oddsratio>
        <expcount>0.15</expcount>
        <count>3</count>
        <size>10</size>
    </kegg>
    <kegg>
        <id>2</id>
        <kegg>05320</kegg>
        <pathname>Autoimmune thyroid disease</pathname>
        <pvalue>0.03</pvalue>
        <oddsratio>24.54</oddsratio>
        <expcount>0.18</expcount>
        <count>3</count>
        <size>12</size>
    </kegg>
</keggs>
```

2.8   The expression data

Finally, the expression data in included in the `<data>` tag. This is a Base64 encoded string, generated by representing each expression value with a single unsigned byte, and then concatenating and Base64 encoding these bytes. This data looks like this:

```
<data>
/Acs1hZT+FDeFGH/29zL+3rK4+sTCvzZzxVgvNrM5Bw8Kfjw2vb6Of8qti943/0QIF8xJy7g2ufB
zMoOqyYA4/ne1KsZ1bH69u3c28zewtQM9G8T5zGK7POszwIM/ub2z9zwHxclIdz3ywC6xtfOySje
DewMklghAfGqzsQtDeHZ3wolAC1GSN+9N+/i5Oz99bTzzRb1QvzHE+Qlr1Ej1U697+AFvtynDeHd
```

```
...
</data>

</xsd:schema>
```

3   Additional information

Please see the ExpressionView homepage at `http://www.unil.ch/cbg/ExpressionView` for more infomation, the schema file and example data files.

References

[1] Sven Bergmann, Jan Ihmels, and Naama Barkai. Iterative signature algorithm for the analysis of large-scale gene expression data. *Physical Review E*, 67:031902, 2003.