Reading GTF files

Wolfgang Kaisers, CBiBs HHU Dusseldorf

April 25, 2019

1 GTF file format

The definition for GTF file format can be found on the UCSC FAQ page `http://genome.ucsc.edu/FAQ/FAQformat.html`.
GTF is based on GFF file format.

1.1 GFF format

The GFF file format defines text file format. Each record is located within one line. GFF defines nine mandatory tab separated columns.

Nr	Name	Type	Content
1	seqname	text	Sequence name
2	source	text	Program which created record
3	feature	text	Type (e.g. exon)
4	start	integer	1-based start position
5	end	integer	Inclusive end position
6	score	text	Number between 0 and 1000 (”.” = empty value)
7	strand	text	’+’ or ’-’ or ’.’
8	frame	text	Number between 0 and 2 or ”.”.
9	group	text	GTF attributes

1.2 GTF extension of GFF

In GTF, the last `group` column is composed of attributes. Each attribute consists of a type / value pair. Attributes are separated by ”; ” (semicolon and one space). `type` and `value` are separated by one space. Values are enclosed in quotation marks (”).

Example

```
gene_id "ENSG00000227232"; transcript_id "ENST00000438504"
```

2 R interface

Import of GTF files inside the refGenome package is done by the `read.gtf` function. `read.gtf` calls `read_gtf` via `.Call`.

3 C Interface

The source of `read_gtf` is located in refGenome.cpp. Inside `read_gtf` a `gtf::gtf_file` C++ object is created which performs line wise parsing.
The obtained result is then copied into a `data.frame`. The R interface to `data.frame` is encapsulated in a `data_frame` C++ object.
The column vectors of the `data.frame` are contained in `atmptr` C++ objects (the name is a modification of the R extptr type for atomic objects.) GFF derived column values are copied by name.
The content of the variable attribute column is added by iteration through the (gtf_attribute) container.

4 The C++ GTF classes

Definition of the `gtf_file` C++ class is located in the 'gtf.h' header file inside the 'gtf' namespace.

4.1 The gtf_file class

The gtf_file class encapsulates the global functionality for parsing GTF files. The main data content is carried by a list of `gff_element` objects and a `gtf_attribute` class.
GTF file content is parsed linewise via getline. A line is parsed by static extraction of the first eight columns.
The last column is extracted using the `gtf_attribute` class.

4.1.1 GTF attributes

Because the GTF format definition does not include number or type of attributes, a parsing algorithm needs to keep an unkown number of values of an unkown number of types. Each data record (line) is identified by a unique integral id.
The decision here was to use hash table implemented by std::unordered_map (included by <unordered_map>). The map uses the attribute type as key value. Therefore only a small number of keys exist. Each map element consists of a list of id / value pairs. A GTF attribute is added by pushing back the id / value pair to the list residing as map-element in the unordered_map.
The values are retrieved by iteration through the unordered_map. Each map element defines a new column in the returned `data.frame`. The stored id values serve as row indices.

An alternative implementation would have been a linked list containing id, type, value triples which could have been passed back to R inside a data.frame with tree columns. Further separation could then be done by sequentially extraction of items for all present types inside R.

Class	Header file	Function
extptr	extptr.h	EXTPTR type for C++ pointer
atmptr	extptr.h	Atomic objects (e.g. INTSXP)
data_frame	data_frame.h	data.frame objects

5 C++ interface for R types

A C++ interface for some R types is included. The basic idea behind this interface is that it simple enough to be contained in a few small header files. We describe three C++ classes which are defined in two header files.

5.1 extptr

The exptr class is designed for usage of external C or C++ pointers. Inside R, external pointers are accessed via EXTPTRSXP types. In order to prevent memory leaks, a finalizer routine needs to be registered.
The extptr class internally uses shared_ptr objects (defined in <memory> header.)
The following example shows how extptr objects can be used.

```
#include "extptr.h"

SEXP use_my_class(SEXP pArg)
{
    extptr<my_class> arg(pArg);
    arg->exec_function();
    return arg;
}
```

5.2 atmptr

The atmptr class defines operators

- operator*
- operator->
- operator[]
- operator SEXP

which allow using these objects in almost the same way as SEXP objects. Only, atmptr variables are typed and there is no need to care about protection.

The class is implemented as template using template specialisations.

5.2.1 Integer vectors

A simple example shows how atmptr objects can be used.

```
#include "extptr.h"

SEXP square(SEXP pArg)
{
    atmptr<int> arg(pArg);

    int i, n = length(arg);
    atmptr<int> pRes(n);

    for(i=0; i < n; ++i)
        pRes[i] = arg[i]^2;

    return pRes;
}
```

5.2.2 String vectors

For strings, the situation is more complicated because string vectors cannot be simply implemented in arrays.

```
#include "extptr.h"

SEXP get_str()
{
    vector<string> v;
    // do fill vector ...

    unsigned i, n = v.size();
    atmptr<char> res(n);

    for(i=0; i < n; ++i)
        res.set(i, v[i]);

    return res;
}
```

5.3 Data frames

The data_frame class implements construction details for R data.frame's.
The following example shows how to create a data.frame object.

```
#include "extptr.h"
#include "data_frame.h"

SEXP create_data_frame()
{
    int nrow = 3, ncol=4;
    data_frame dfr(nrow, ncol);

    // Add ID - column ("id")
    dfr.addIdColumn();
    atmptr<char> type(nrow);
    // fill types ....
    dfr.addColumn(type, "type");

    atmptr<int> values(nrow);
    // fill values ...
    dfr.addColumn(values, "value")

    return dfr;
}
```

The data_frame constructor adds row.names consisting of consecutive integer numbers starting at 1.