service-ars documentation

Release 0.1

Michael Halagan

January 30, 2017

The Gene Feature Enumeration (GFE) Submission service provides an API for converting raw sequence data to GFE. It provides both a RESTful API and a simple user interface for converting raw sequence data to GFE results. Sequences can be submitted one at a time or as a fasta file. This service uses feature service for encoding the raw sequence data and annotation pipeline for aligning the raw sequence data. A public version of this service is available for use at gfe.b12x.org.

The code is open source, and available on GitHub.

CHAPTER 1

Overview

GFE is described in more detail on the IGDAWG website.

1.1 Paper

In 2015 the paper introducing GFE ("A gene feature enumeration approach for describing HLA allele polymorphism") was published by Steve Mack in Human Immunology. The paper describes the GFE notation, its applicaitons, and the development of the feature service.

1.2 GFE Nomenclature

Note: Currently, the GFE nomenclature returned from the service does not represent partial sequences. In the next release we intend to represent this as an *aligned* percent for each *feature* to the reference sequence. This is similar to the *aligned* value returned for the full sequence.

The GFE allele name is created by joining the *feature* accesion numbers. Each accession number is generated by making a RESTful call to the feature service with the sequence and it's corresponding *feature*. If the sequence has been observed before, the the feature service will return the previously generated accession number, otherwise it will generate a unique accession number for that feature and return it. The GFE allele **HLA-Aw1-1-7-20-10-32-7-1-1-1-6-1-5-3-5-1-1** is broken up into the following parts:

Feature	Accession Number
five_prime_UTR	1
Exon 1	1
Intron 1	7
Exon 2	20
Intron 2	10
Exon 3	32
Intron 3	7
Exon 4	1
Intron 4	1
Exon 5	1
Intron 5	6
Exon 6	1
Intron 6	5
Exon 7	3
Intron 7	5
Exon 8	1
three_prime_UTR	1

In the example above all of the features for HLA-A were found in the sequence. If a sequence does not contain a particular feature, then the feature accession number will be zero. The raw sequence is broken up into each of its features using the annotation pipeline.

CHAPTER 2

RESTful API

Note: Each of the APIs described below can be tested out using the Swagger API interface.

The *structures* parameter is for returning each part of the GFE allele as a *structure object*. By default it will always return the full structure, but you may want it to only return the GFE annotation. The *retry* parameter defines how many times you want the GFE service to retry a call to the feature service. The default is *six* and should only be changed for debugging purposes. Open an a issue on github with any suggested changes you have or any issues you notice with the service.

2.1 POST /gfe

Tip: I suggest always using the *verbose* parameter for more detailed documentation of any potentail errors.

Turning sequence data to GFE annotation can be done by using the gfe API. If you have a large number of sequences you want to convert to GFE annotation then refer to the *fasta* or *hml* APIs. At the very minimum you only have to provide a sequence and a locus. Below are the input parameters for the gfe API:

Here is an example of a JSON object that can be posted to the gfe API:

```
    {
      "locus": "HLA-A",
      "verbose":1,
      "sequence": "TCCCCAGACGCCGAGGATGGCCGTCATGGCGCCCCGAACCCTCCTCCTGCTACTCTCGGGGGGCCCTGGCCCTGACCCAGAC
    }
```

The reponse from the API will either be a GFE JSON object or an *error object*. Here is the GFE object model: The reponse will always contain a *fullgene* object, which contains the accession number for the full gene sequence. This accession number can be used to retrieve the sequence from the *feature-service*. The *aligned* number represents what

percent of the submitted sequence was able to be aligned to the reference. If there is a large insertion or deletion in the submitted sequence, then the *aligned* value should reflect that.

Here is the JSON that would be returned from posting the above JSON to the gfe API:


```
{
  "aligned": "1.000",
  "fullgene": {
    "accession": "1",
    "rank": "1",
    "sequence": "TCCCCAGACGCCGAGGATGGCCGTCATGGCGCCCCGAACCCTCCTCCTGCTACTCTCGGGGGCCCTGGCCCTGACCCAGACCTC
    "term": "gene"
  },
  "gfe": "HLA-Aw1-1-7-20-10-32-7-1-1-1-6-1-5-3-5-1-1",
  "locus": "HLA-A",
  "log": [
    "2017/01/20 19:57:10 INFO> GFE.pm:1317 GFE::checkFile - File is valid",
    "2017/01/20 19:57:10 INFO> GFE.pm:1331 GFE::checkFile - File is valid for fasta type",
    "2017/01/20 19:57:16 INFO> GFE.pm:1209 GFE::checkExitStatus - Alignment ran successfully",
    "2017/01/20 19:57:16 INFO> Annotate.pm:234 GFE::Annotate::alignment_file - Alignment file: /opt/h
    "2017/01/20 19:57:16 INFO> Annotate.pm:234 GFE::Annotate::alignment_file - Alignment file: /opt/h
    "2017/01/20 19:57:16 INFO> GFE.pm:1127 GFE::checkAlignmentFile - Generated alignment file: /opt/h
    "2017/01/20 19:57:16 INFO> Annotate.pm:234 GFE::Annotate::alignment_file - Alignment file: /opt/h
    "2017/01/20 19:57:16 INFO> GFE.pm:1010 GFE::checkResults - Successfully loaded results",
    "2017/01/20 19:57:16 INFO> GFE.pm:1154 GFE::checkGfe - Generated GFE: HLA-Aw1-1-7-20-10-32-7-1-1-
  ],
  "structure": [
    {
      "accession": "1",
      "rank": "1",
      "sequence": "TCCCCAGACGCCGAGG",
      "term": "five_prime_UTR"
    },
    {
      "accession": "1",
      "rank": "1",
      "sequence": "ATGGCCGTCATGGCGCCCCGAACCCTCCTCCTGCTACTCTCGGGGGCCCTGGCCCTGACCCAGACCTGGGCGG",
      "term": "exon"
    },
    {
      "accession": "7",
      "rank": "1",
      "sequence": "GTGAGTGCGGGGTCGGGAGGGAAACCGCCTCTGCGGGGAGAAGCAAGGGGCCCTCCTGGCGGGGGCGCAGGACCGGGGGAGC
      "term": "intron"
    },
    {
      "accession": "20",
      "rank": "2",
      "sequence": "GCTCCCACTCCATGAGGTATTTCTTCACATCCGTGTCCCGGCCCGGCCGCGGGGAGCCCCGCTTCATCGCCGTGGGCTACGT
      "term": "exon"
    },
    {
      "accession": "10",
      "rank": "2",
      "sequence": "GTGAGTGACCCCGGCCGGGGGCGCAGGTCAGGACCCCTCATCCCCCACGGACGGGCCAGGTCGCCCACAGTCTCCGGGTCCC
      "term": "intron"
    },
    {
      "accession": "32",
```

```
    "rank": "3",
    "sequence": "GTTCTCACACCATCCAGATAATGTATGGCTGCGACGTGGGGTCGGACGGGCGCTTCCTCCGCGGGTACCGGCAGGACGCCTA
    "term": "exon"
  },
  {
    "accession": "7",
    "rank": "3",
    "sequence": "GTACCAGGGGCCACGGGGCGCCTCCCTGATCGCCTGTAGATCTCCCGGGCTGGCCTCCCACAAGGAGGGGAGACAATTGGGA
    "term": "intron"
  },
  {
    "accession": "1",
    "rank": "4",
    "sequence": "ACCCCCCCAAGACACATATGACCCACCACCCCATCTCTGACCATGAGGCCACCCTGAGGTGCTGGGCCCTGGGCTTCTACCC
    "term": "exon"
  },
  {
    "accession": "1",
    "rank": "4",
    "sequence": "GTAAGGAGGGAGATGGGGGTGTCATGTCTCTTAGGGAAAGCAGGAGCCTCTCTGGAGACCTTTAGCAGGGTCAGGGCCCCTC
    "term": "intron"
  },
  {
    "accession": "1",
    "rank": "5",
    "sequence": "AGCTGTCTTCCCAGCCCACCATCCCCATCGTGGGCATCATTGCTGGCCTGGTTCTCCTTGGAGCTGTGATCACTGGAGCTGT
    "term": "exon"
  },
  {
    "accession": "6",
    "rank": "5",
    "sequence": "GTGGAGAAGGGGTGAAGGGTGGGGTCTGAGATTTCTTGTCTCACTGAGGGTTCCAAGCCCCAGCTAGAAATGTGCCCTGTCT
    "term": "intron"
  },
  {
    "accession": "1",
    "rank": "6",
    "sequence": "ATAGAAAAGGAGGGAGTTACACTCAGGCTGCAA",
    "term": "exon"
  },
  {
    "accession": "5",
    "rank": "6",
    "sequence": "GTAAGTATGAAGGAGGCTGATGCCTGAGGTCCTTGGGATATTGTGTTTGGGAGCCCATGGGGGAGCTCACCCACCTCACAAT
    "term": "intron"
  },
  {
    "accession": "3",
    "rank": "7",
    "sequence": "GCAGTGACAGTGCCCAGGGCTCTGATGTGTCCCTCACAGCTTGTAAAG",
    "term": "exon"
  },
  {
    "accession": "5",
    "rank": "7",
    "sequence": "GTGAGAGCTTGGAGGACCTAATGTGTGTTGGGTGTTGGGCGGAACAGTGGACACAGCTGTGCTATGGGGTTTCTTTGCATTC
    "term": "intron"
  },
```

```
    {
      "accession": "1",
      "rank": "8",
      "sequence": "TGTGA",
      "term": "exon"
    },
    {
      "accession": "1",
      "rank": "8",
      "sequence": "GACAGCTGCCTTGTGTGGGACTGAG",
      "term": "three_prime_UTR"
    }
  ],
  "version": "1.1.1"
}
```


You can generate the above JSON by running the following `curl` command:

```
curl --header "Content-type: application/JSON" --request POST \
--data '{"locus":"HLA-A","verbose":1,sequence":"TCCCCAGACGCCGAGGATGGCCGTCATGGCGCCCCGAACCCTCCTCCTGCTAC
http://gfe.b12x.org/gfe
```

2.2 POST /sequence

Turning GFE annotations back to the full sequence can be done using the sequence API. When submitting a sequence, if the *aligned* value is less than 1 then you will not be able to get the complete sequence back from the GFE annotation. The *fullgene* accession number returned from the gfe API can be used to get back the full sequence from the feature service. Below are the input parameters for the sequence API:

At the very minimum you only have you provide a gfe annotation and a locus. As with the gfe API, the structures of the GFE are returned by default. Here is an example of a JSON object that can be posted to the sequence API:


```
{
  "gfe": "HLA-Aw1-1-7-20-10-32-7-1-1-1-6-1-5-3-5-1-0",
  "locus": "HLA-A",
  "verbose": 1
}
```


The reponse from the API will either be a Sequence JSON object or an *error object*. The reponse model for the sequence API is as follows:

Here is the JSON that would be returned from posting the above JSON to the sequence API:


```
{
  "log": [
    "2017/01/20 19:34:30 INFO> GFE.pm:1154 GFE::checkGfe - Generated GFE: HLA-Aw1-1-7-20-10-32-7-1-1-
    "2017/01/20 19:34:30 WARN> GFE.pm:750 GFE::getSequence - Accession is not defined       - GFE:
  ],
  "sequence": "TCCCCAGACGCCGAGGATGGCCGTCATGGCGCCCCGAACCCTCCTCCTGCTACTCTCGGGGGCCCTGGCCCTGACCCAGACCTGGG
  "structure": [
    {
      "accession": "1",
      "locus": "HLA-A",
      "rank": "1",
      "sequence": "TCCCCAGACGCCGAGG",
      "term": "five_prime_UTR"
    },
```

```
  {
    "accession": "1",
    "locus": "HLA-A",
    "rank": "1",
    "sequence": "ATGGCCGTCATGGCGCCCCGAACCCTCCTCCTGCTACTCTCGGGGGCCCTGGCCCTGACCCAGACCTGGGCGG",
    "term": "exon"
  },
  {
    "accession": "7",
    "locus": "HLA-A",
    "rank": "1",
    "sequence": "GTGAGTGCGGGGTCGGGAGGGAAACCGCCTCTGCGGGGAGAAGCAAGGGGCCCTCCTGGCGGGGGCGCAGGACCGGGGGAGC
    "term": "intron"
  },
  {
    "accession": "20",
    "locus": "HLA-A",
    "rank": "2",
    "sequence": "GCTCCCACTCCATGAGGTATTTCTTCACATCCGTGTCCCGGCCCGGCCGCGGGGAGCCCCGCTTCATCGCCGTGGGCTACGT
    "term": "exon"
  },
  {
    "accession": "10",
    "locus": "HLA-A",
    "rank": "2",
    "sequence": "GTGAGTGACCCCGGCCGGGGGCGCAGGTCAGGACCCCTCATCCCCCACGGACGGGCCAGGTCGCCCACAGTCTCCGGGTCCC
    "term": "intron"
  },
  {
    "accession": "32",
    "locus": "HLA-A",
    "rank": "3",
    "sequence": "GTTCTCACACCATCCAGATAATGTATGGCTGCGACGTGGGGTCGGACGGGCGCTTCCTCCGCGGGTACCGGCAGGACGCCTA
    "term": "exon"
  },
  {
    "accession": "7",
    "locus": "HLA-A",
    "rank": "3",
    "sequence": "GTACCAGGGGCCACGGGGCGCCTCCCTGATCGCCTGTAGATCTCCCGGGCTGGCCTCCCACAAGGAGGGGAGACAATTGGGA
    "term": "intron"
  },
  {
    "accession": "1",
    "locus": "HLA-A",
    "rank": "4",
    "sequence": "ACCCCCCCAAGACACATATGACCCACCACCCCATCTCTGACCATGAGGCCACCCTGAGGTGCTGGGCCCTGGGCTTCTACCC
    "term": "exon"
  },
  {
    "accession": "1",
    "locus": "HLA-A",
    "rank": "4",
    "sequence": "GTAAGGAGGGAGATGGGGGTGTCATGTCTCTTAGGGAAAGCAGGAGCCTCTCTGGAGACCTTTAGCAGGGTCAGGGCCCCTC
    "term": "intron"
  },
  {
    "accession": "1",
```

```
      "locus": "HLA-A",
      "rank": "5",
      "sequence": "AGCTGTCTTCCCAGCCCACCATCCCCATCGTGGGCATCATTGCTGGCCTGGTTCTCCTTGGAGCTGTGATCACTGGAGCTGT
      "term": "exon"
    },
    {
      "accession": "6",
      "locus": "HLA-A",
      "rank": "5",
      "sequence": "GTGGAGAAGGGGTGAAGGGTGGGGTCTGAGATTTCTTGTCTCACTGAGGGTTCCAAGCCCCAGCTAGAAATGTGCCCTGTCT
      "term": "intron"
    },
    {
      "accession": "1",
      "locus": "HLA-A",
      "rank": "6",
      "sequence": "ATAGAAAAGGAGGGAGTTACACTCAGGCTGCAA",
      "term": "exon"
    },
    {
      "accession": "5",
      "locus": "HLA-A",
      "rank": "6",
      "sequence": "GTAAGTATGAAGGAGGCTGATGCCTGAGGTCCTTGGGATATTGTGTTTGGGAGCCCATGGGGGAGCTCACCCACCTCACAAT
      "term": "intron"
    },
    {
      "accession": "3",
      "locus": "HLA-A",
      "rank": "7",
      "sequence": "GCAGTGACAGTGCCCAGGGCTCTGATGTGTCCCTCACAGCTTGTAAAG",
      "term": "exon"
    },
    {
      "accession": "5",
      "locus": "HLA-A",
      "rank": "7",
      "sequence": "GTGAGAGCTTGGAGGACCTAATGTGTGTTGGGTGTTGGGCGGAACAGTGGACACAGCTGTGCTATGGGGTTTCTTTGCATTC
      "term": "intron"
    },
    {
      "accession": "1",
      "locus": "HLA-A",
      "rank": "8",
      "sequence": "TGTGA",
      "term": "exon"
    }
  ],
  "version": "1.0.7"
}
```


You can generate the above JSON by running the following `curl` command:

```
curl --header "Content-type: application/JSON" --request POST \
--data '{"locus":"HLA-A","gfe":"HLA-Aw1-1-7-20-10-32-7-1-1-1-6-1-5-3-5-1-0"}' \
http://gfe.b12x.org/sequence
```

2.3 POST /fasta

Converting sequences in a fasta file to GFE annotation can be done using the fasta API. All of the sequences in the fasta file need to be from the same locus. Below are the input parameters for the fasta API:

Here is an example of JSON that can be posted to the fasta API:

```
{
  "file": "GFE_Submission/t/resources/fastatest1.fasta",
  "locus": "HLA-A",
  "verbose": 1
}
```

The reponse from the API will either be a SubjectData object or an *error object*. The GFE reponse object model is as follows: The *Subject* object id will be whatever the fasta headers are. Here is the JSON that would be returned from posting the above JSON to the fasta API:

```
{
   "log" : [
      "2017/01/22 21:06:37 INFO> GFE.pm:1317 GFE::checkFile - File is valid",
      "2017/01/22 21:06:37 INFO> GFE.pm:1331 GFE::checkFile - File is valid for fasta type",
      "2017/01/22 21:06:37 INFO> Annotate.pm:190 GFE::Annotate::setFastaFile - Input file: public/dow
      "2017/01/22 21:06:51 INFO> GFE.pm:1209 GFE::checkExitStatus - Alignment ran successfully",
      "2017/01/22 21:06:51 INFO> Annotate.pm:234 GFE::Annotate::alignment_file - Alignment file: /Use
      "2017/01/22 21:06:51 INFO> Annotate.pm:234 GFE::Annotate::alignment_file - Alignment file: /Use
      "2017/01/22 21:06:51 INFO> GFE.pm:1127 GFE::checkAlignmentFile - Generated alignment file: /Use
      "2017/01/22 21:06:56 INFO> Annotate.pm:234 GFE::Annotate::alignment_file - Alignment file: /Use
      "2017/01/22 21:06:56 INFO> GFE.pm:1010 GFE::checkResults - Successfully loaded results",
      "2017/01/22 21:06:56 INFO> GFE.pm:1154 GFE::checkGfe - Generated GFE: HLA-Aw1-1-7-20-10-32-7-1-
      "2017/01/22 21:06:56 INFO> GFE.pm:1154 GFE::checkGfe - Generated GFE: HLA-Aw1-68-7-20-10-32-7-1
      "2017/01/22 21:06:56 INFO> GFE.pm:1154 GFE::checkGfe - Generated GFE: HLA-Aw1-1-7-20-10-32-7-1-
      "2017/01/22 21:06:56 INFO> GFE.pm:1154 GFE::checkGfe - Generated GFE: HLA-Aw1-1-7-20-10-32-7-1-
   ],
   "subjects" : [
      {
         "id" : "HLA-A*01:01:01",
         "typingData" : [
            {
               "locus" : "HLA-A",
               "typing" : [
                  {
                     "gfe" : "HLA-Aw1-1-7-20-10-32-7-1-1-1-6-1-5-3-5-1-1",
                     "structure" : [
                        {
                           "accession" : "1",
                           "rank" : "1",
                           "sequence" : "TCCCCAGACGCCGAGG",
                           "term" : "five_prime_UTR"
                        },
                        {
                           "accession" : "1",
                           "rank" : "1",
                           "sequence" : "ATGGCCGTCATGGCGCCCCGAACCCTCCTCCTGCTACTCTCGGGGGCCCTGGCCCTGACC
                           "term" : "exon"
                        },
                        {
                           "accession" : "7",
                           "rank" : "1",
```

```
            "sequence" : "GTGAGTGCGGGGTCGGGAGGGAAACCGCCTCTGCGGGGAGAAGCAAGGGGCCCTCCTGGG
            "term" : "intron"
        },
        {
            "accession" : "20",
            "rank" : "2",
            "sequence" : "GCTCCCACTCCATGAGGTATTTCTTCACATCCGTGTCCCGGCCCGGCCGCGGGGAGCCCC
            "term" : "exon"
        },
        {
            "accession" : "10",
            "rank" : "2",
            "sequence" : "GTGAGTGACCCCGGCCGGGGGCGCAGGTCAGGACCCCTCATCCCCCACGGACGGGCCAGC
            "term" : "intron"
        },
        {
            "accession" : "32",
            "rank" : "3",
            "sequence" : "GTTCTCACACCATCCAGATAATGTATGGCTGCGACGTGGGGTCGGACGGGCGCTTCCTCC
            "term" : "exon"
        },
        {
            "accession" : "7",
            "rank" : "3",
            "sequence" : "GTACCAGGGGCCACGGGGCGCCTCCCTGATCGCCTGTAGATCTCCCGGGCTGGCCTCCCA
            "term" : "intron"
        },
        {
            "accession" : "1",
            "rank" : "4",
            "sequence" : "ACCCCCCCAAGACACATATGACCCACCACCCCATCTCTGACCATGAGGCCACCCTGAGGT
            "term" : "exon"
        },
        {
            "accession" : "1",
            "rank" : "4",
            "sequence" : "GTAAGGAGGGAGATGGGGGTGTCATGTCTCTTAGGGAAAGCAGGAGCCTCTCTGGAGACC
            "term" : "intron"
        },
        {
            "accession" : "1",
            "rank" : "5",
            "sequence" : "AGCTGTCTTCCCAGCCCACCATCCCCATCGTGGGCATCATTGCTGGCCTGGTTCTCCTTC
            "term" : "exon"
        },
        {
            "accession" : "6",
            "rank" : "5",
            "sequence" : "GTGGAGAAGGGGTGAAGGGTGGGGTCTGAGATTTCTTGTCTCACTGAGGGTTCCAAGCCC
            "term" : "intron"
        },
        {
            "accession" : "1",
            "rank" : "6",
            "sequence" : "ATAGAAAAGGAGGGAGTTACACTCAGGCTGCAA",
            "term" : "exon"
        },
        {
```

```
                        "accession" : "5",
                        "rank" : "6",
                        "sequence" : "GTAAGTATGAAGGAGGCTGATGCCTGAGGTCCTTGGGATATTGTGTTTGGGAGCCCATG
                        "term" : "intron"
                     },
                     {
                        "accession" : "3",
                        "rank" : "7",
                        "sequence" : "GCAGTGACAGTGCCCAGGGCTCTGATGTGTCCCTCACAGCTTGTAAAG",
                        "term" : "exon"
                     },
                     {
                        "accession" : "5",
                        "rank" : "7",
                        "sequence" : "GTGAGAGCTTGGAGGACCTAATGTGTGTTGGGTGTTGGGCGGAACAGTGGACACAGCTG
                        "term" : "intron"
                     },
                     {
                        "accession" : "1",
                        "rank" : "8",
                        "sequence" : "TGTGA",
                        "term" : "exon"
                     },
                     {
                        "accession" : "1",
                        "rank" : "8",
                        "sequence" : "GACAGCTGCCTTGTGTGGGACTGAG",
                        "term" : "three_prime_UTR"
                     }
                  ]
               }
            ]
         }
      ]
   },
   {
      "id" : "HLA-A*01:01:03",
      "typingData" : [
         {
            "locus" : "HLA-A",
            "typing" : [
               {
                  "gfe" : "HLA-Aw1-68-7-20-10-32-7-1-1-1-6-1-5-3-5-1-1",
                  "structure" : [
                     {
                        "accession" : "1",
                        "rank" : "1",
                        "sequence" : "TCCCCAGACGCCGAGG",
                        "term" : "five_prime_UTR"
                     },
                     {
                        "accession" : "68",
                        "rank" : "1",
                        "sequence" : "ATGGCCGTCATGGCCCAGACCTGGGCGG",
                        "term" : "exon"
                     },
                     {
                        "accession" : "7",
```

```
            "rank" : "1",
            "sequence" : "GTGAGTGCGGGGTCGGGAGGGAAACCGCCTCTGCGGGGAGAAGCAAGGGGCCCTCCTGGG
            "term" : "intron"
        },
        {
            "accession" : "20",
            "rank" : "2",
            "sequence" : "GCTCCCACTCCATGAGGTATTTCTTCACATCCGTGTCCCGGCCCGGCCGCGGGGAGCCCC
            "term" : "exon"
        },
        {
            "accession" : "10",
            "rank" : "2",
            "sequence" : "GTGAGTGACCCCGGCCGGGGGCGCAGGTCAGGACCCCTCATCCCCCACGGACGGGCCAGC
            "term" : "intron"
        },
        {
            "accession" : "32",
            "rank" : "3",
            "sequence" : "GTTCTCACACCATCCAGATAATGTATGGCTGCGACGTGGGGTCGGACGGGCGCTTCCTCC
            "term" : "exon"
        },
        {
            "accession" : "7",
            "rank" : "3",
            "sequence" : "GTACCAGGGGCCACGGGGCGCCTCCCTGATCGCCTGTAGATCTCCCGGGCTGGCCTCCCA
            "term" : "intron"
        },
        {
            "accession" : "1",
            "rank" : "4",
            "sequence" : "ACCCCCCCAAGACACATATGACCCACCACCCCATCTCTGACCATGAGGCCACCCTGAGGT
            "term" : "exon"
        },
        {
            "accession" : "1",
            "rank" : "4",
            "sequence" : "GTAAGGAGGGAGATGGGGGTGTCATGTCTCTTAGGGAAAGCAGGAGCCTCTCTGGAGACC
            "term" : "intron"
        },
        {
            "accession" : "1",
            "rank" : "5",
            "sequence" : "AGCTGTCTTCCCAGCCCACCATCCCCATCGTGGGCATCATTGCTGGCCTGGTTCTCCTTC
            "term" : "exon"
        },
        {
            "accession" : "6",
            "rank" : "5",
            "sequence" : "GTGGAGAAGGGGTGAAGGGTGGGGTCTGAGATTTCTTGTCTCACTGAGGGTTCCAAGCCC
            "term" : "intron"
        },
        {
            "accession" : "1",
            "rank" : "6",
            "sequence" : "ATAGAAAAGGAGGGAGTTACACTCAGGCTGCAA",
            "term" : "exon"
        },
```

```
                        {
                           "accession" : "5",
                           "rank" : "6",
                           "sequence" : "GTAAGTATGAAGGAGGCTGATGCCTGAGGTCCTTGGGATATTGTGTTTGGGAGCCCATGC
                           "term" : "intron"
                        },
                        {
                           "accession" : "3",
                           "rank" : "7",
                           "sequence" : "GCAGTGACAGTGCCCAGGGCTCTGATGTGTCCCTCACAGCTTGTAAAG",
                           "term" : "exon"
                        },
                        {
                           "accession" : "5",
                           "rank" : "7",
                           "sequence" : "GTGAGAGCTTGGAGGACCTAATGTGTGTTGGGTGTTGGGCGGAACAGTGGACACAGCTGT
                           "term" : "intron"
                        },
                        {
                           "accession" : "1",
                           "rank" : "8",
                           "sequence" : "TGTGA",
                           "term" : "exon"
                        },
                        {
                           "accession" : "1",
                           "rank" : "8",
                           "sequence" : "GACAGCTGCCTTGTGTGGGACTGAG",
                           "term" : "three_prime_UTR"
                        }
                     ]
                  }
               ]
            }
         ]
      },
      {
         "id" : "HLA-A*01:01:04",
         "typingData" : [
            {
               "locus" : "HLA-A",
               "typing" : [
                  {
                     "gfe" : "HLA-Aw1-1-7-20-10-32-7-1-1-1-6-1-5-3-5-1-1",
                     "structure" : [
                        {
                           "accession" : "1",
                           "rank" : "1",
                           "sequence" : "TCCCCAGACGCCGAGG",
                           "term" : "five_prime_UTR"
                        },
                        {
                           "accession" : "1",
                           "rank" : "1",
                           "sequence" : "ATGGCCGTCATGGCGCCCCGAACCCTCCTCCTGCTACTCTCGGGGGCCCTGGCCCTGACC
                           "term" : "exon"
                        },
                        {
```

```
            "accession" : "7",
            "rank" : "1",
            "sequence" : "GTGAGTGCGGGGTCGGGAGGGAAACCGCCTCTGCGGGGAGAAGCAAGGGGCCCTCCTGGG
            "term" : "intron"
         },
         {
            "accession" : "20",
            "rank" : "2",
            "sequence" : "GCTCCCACTCCATGAGGTATTTCTTCACATCCGTGTCCCGGCCCGGCCGCGGGGAGCCCC
            "term" : "exon"
         },
         {
            "accession" : "10",
            "rank" : "2",
            "sequence" : "GTGAGTGACCCCGGCCGGGGGCGCAGGTCAGGACCCCTCATCCCCCACGGACGGGCCAGC
            "term" : "intron"
         },
         {
            "accession" : "32",
            "rank" : "3",
            "sequence" : "GTTCTCACACCATCCAGATAATGTATGGCTGCGACGTGGGGTCGGACGGGCGCTTCCTCC
            "term" : "exon"
         },
         {
            "accession" : "7",
            "rank" : "3",
            "sequence" : "GTACCAGGGGCCACGGGGCGCCTCCCTGATCGCCTGTAGATCTCCCGGGCTGGCCTCCCA
            "term" : "intron"
         },
         {
            "accession" : "1",
            "rank" : "4",
            "sequence" : "ACCCCCCCAAGACACATATGACCCACCACCCCATCTCTGACCATGAGGCCACCCTGAGGT
            "term" : "exon"
         },
         {
            "accession" : "1",
            "rank" : "4",
            "sequence" : "GTAAGGAGGGAGATGGGGGTGTCATGTCTCTTAGGGAAAGCAGGAGCCTCTCTGGAGACC
            "term" : "intron"
         },
         {
            "accession" : "1",
            "rank" : "5",
            "sequence" : "AGCTGTCTTCCCAGCCCACCATCCCCATCGTGGGCATCATTGCTGGCCTGGTTCTCCTTC
            "term" : "exon"
         },
         {
            "accession" : "6",
            "rank" : "5",
            "sequence" : "GTGGAGAAGGGGTGAAGGGTGGGGTCTGAGATTTCTTGTCTCACTGAGGGTTCCAAGCCC
            "term" : "intron"
         },
         {
            "accession" : "1",
            "rank" : "6",
            "sequence" : "ATAGAAAAGGAGGGAGTTACACTCAGGCTGCAA",
            "term" : "exon"
```

```
                        },
                        {
                           "accession" : "5",
                           "rank" : "6",
                           "sequence" : "GTAAGTATGAAGGAGGCTGATGCCTGAGGTCCTTGGGATATTGTGTTTGGGAGCCCATGC
                           "term" : "intron"
                        },
                        {
                           "accession" : "3",
                           "rank" : "7",
                           "sequence" : "GCAGTGACAGTGCCCAGGGCTCTGATGTGTCCCTCACAGCTTGTAAAG",
                           "term" : "exon"
                        },
                        {
                           "accession" : "5",
                           "rank" : "7",
                           "sequence" : "GTGAGAGCTTGGAGGACCTAATGTGTGTTGGGTGTTGGGCGGAACAGTGGACACAGCTGT
                           "term" : "intron"
                        },
                        {
                           "accession" : "1",
                           "rank" : "8",
                           "sequence" : "TGTGA",
                           "term" : "exon"
                        },
                        {
                           "accession" : "1",
                           "rank" : "8",
                           "sequence" : "GACAGCTGCCTTGTGTGGGACTGAG",
                           "term" : "three_prime_UTR"
                        }
                     ]
                  }
               ]
            }
         ]
      },
      {
         "id" : "HLA-A*01:01:02",
         "typingData" : [
            {
               "locus" : "HLA-A",
               "typing" : [
                  {
                     "gfe" : "HLA-Aw1-1-7-20-10-32-7-1-1-1-6-1-5-3-5-1-1",
                     "structure" : [
                        {
                           "accession" : "1",
                           "rank" : "1",
                           "sequence" : "TCCCCAGACGCCGAGG",
                           "term" : "five_prime_UTR"
                        },
                        {
                           "accession" : "1",
                           "rank" : "1",
                           "sequence" : "ATGGCCGTCATGGCGCCCCGAACCCTCCTCCTGCTACTCTCGGGGGCCCTGGCCCTGACC
                           "term" : "exon"
                        },
```

```
                    {
                        "accession" : "7",
                        "rank" : "1",
                        "sequence" : "GTGAGTGCGGGGTCGGGAGGGAAACCGCCTCTGCGGGGAGAAGCAAGGGGCCCTCCTGGG
                        "term" : "intron"
                    },
                    {
                        "accession" : "20",
                        "rank" : "2",
                        "sequence" : "GCTCCCACTCCATGAGGTATTTCTTCACATCCGTGTCCCGGCCCGGCCGCGGGGAGCCCC
                        "term" : "exon"
                    },
                    {
                        "accession" : "10",
                        "rank" : "2",
                        "sequence" : "GTGAGTGACCCCGGCCGGGGGCGCAGGTCAGGACCCCTCATCCCCCACGGACGGGCCAGC
                        "term" : "intron"
                    },
                    {
                        "accession" : "32",
                        "rank" : "3",
                        "sequence" : "GTTCTCACACCATCCAGATAATGTATGGCTGCGACGTGGGGTCGGACGGGCGCTTCCTCC
                        "term" : "exon"
                    },
                    {
                        "accession" : "7",
                        "rank" : "3",
                        "sequence" : "GTACCAGGGGCCACGGGGCGCCTCCCTGATCGCCTGTAGATCTCCCGGGCTGGCCTCCCA
                        "term" : "intron"
                    },
                    {
                        "accession" : "1",
                        "rank" : "4",
                        "sequence" : "ACCCCCCCCAAGACACATATGACCCACCACCCCATCTCTGACCATGAGGCCACCCTGAGGT
                        "term" : "exon"
                    },
                    {
                        "accession" : "1",
                        "rank" : "4",
                        "sequence" : "GTAAGGAGGGAGATGGGGGTGTCATGTCTCTTAGGGAAAGCAGGAGCCTCTCTGGAGACC
                        "term" : "intron"
                    },
                    {
                        "accession" : "1",
                        "rank" : "5",
                        "sequence" : "AGCTGTCTTCCCAGCCCACCATCCCCATCGTGGGCATCATTGCTGGCCTGGTTCTCCTTC
                        "term" : "exon"
                    },
                    {
                        "accession" : "6",
                        "rank" : "5",
                        "sequence" : "GTGGAGAAGGGGTGAAGGGTGGGGTCTGAGATTTCTTGTCTCACTGAGGGTTCCAAGCCC
                        "term" : "intron"
                    },
                    {
                        "accession" : "1",
                        "rank" : "6",
                        "sequence" : "ATAGAAAAGGAGGGAGTTACACTCAGGCTGCAA",
```

```
                            "term" : "exon"
                        },
                        {
                            "accession" : "5",
                            "rank" : "6",
                            "sequence" : "GTAAGTATGAAGGAGGCTGATGCCTGAGGTCCTTGGGATATTGTGTTTGGGAGCCCATGC
                            "term" : "intron"
                        },
                        {
                            "accession" : "3",
                            "rank" : "7",
                            "sequence" : "GCAGTGACAGTGCCCAGGGCTCTGATGTGTCCCTCACAGCTTGTAAAG",
                            "term" : "exon"
                        },
                        {
                            "accession" : "5",
                            "rank" : "7",
                            "sequence" : "GTGAGAGCTTGGAGGACCTAATGTGTGTTGGGTGTTGGGCGGAACAGTGGACACAGCTGT
                            "term" : "intron"
                        },
                        {
                            "accession" : "1",
                            "rank" : "8",
                            "sequence" : "TGTGA",
                            "term" : "exon"
                        },
                        {
                            "accession" : "1",
                            "rank" : "8",
                            "sequence" : "GACAGCTGCCTTGTGTGGGACTGAG",
                            "term" : "three_prime_UTR"
                        }
                    ]
                }
            ]
        }
    ],
    "version" : "1.0.7"
}
```


You can generate the above JSON by running the following `curl` command:

```
curl -F "verbose=1" -F "locus=HLA-A" \
-F "file=@GFE_Submission/t/resources/fastatest1.fasta" \
http://gfe.b12x.org/fasta
```

2.4 POST /hml

Converting sequences in a HML file to GFE annotation can be done using the hml API. Below are the input parameters for the hml API:

Here is an example of a JSON object that can be posted to the hml API:

```
{
  "file": "GFE_Submission/t/resources/hmltest1.HML",
  "type":"JSON",
  "verbose": 1
}
```

The response from the hml API is either a *SubjectData object* or an *error object*. The *Subject* object id will be the subject ids in the HML file. The *type* parameter allows you to specify either JSON or HML output. The default for *type* is JSON. If you chose HML as the output then the GFE annotation will be included within *typing element* in the HML file as a new *allele-assignment*.

2.5 POST /flowhml

Converting sequences in a HML file to GFE annotation can be done using the flowhml API. This API does the same as *hml* API but it runs significantly faster and the *structure object* for each feature will not be returned. The response from the flowhml API is either a *SubjectData object* or an *error object*. The *Subject* object id will be the subject ids in the HML file. The *type* parameter allows you to specify either JSON or HML output. The default for *type* is JSON. If you chose HML as the output then the GFE annotation will be included within *typing element* in the HML file as a new *allele-assignment*.

2.6 Error Object

The model for the error reponse is as follows:

CHAPTER 3

Clients

3.1 Swagger

> "*Swagger is a specification and framework implementation*
> *for describing, producing, consuming, and visualizing RESTful web services.*"

Creating new clients is easy when you use the *swagger.yaml* and Swagger autogeneration. You can autogenerate clients using the Swagger editor or by locallying running the Swagger autogeneration tools. I'll go through the steps for creating a new client using the GFE service *swagger.yaml* and the *Swagger editor*.

1. Go to the Swagger editor.
2. Import the GFE service *swagger.yaml* by either pasting the text or by clicking *File* and then *Import URL*. Import the URL for the raw text of the *swagger.yaml*.
3. Click on *Generate Client* and then select the language you would like to use.

3.2 R Client

Note: The R client hasn't been updated yet to include the nextflow APIs. Open the GfeClient.R script in the client-R directory to see all of the available functions.

The R client is available in the *client-R* directory in the github repositiory. Here are a few examples of installing and using the R client:

```
if (!is.installed('gfeClient')){
    library(devtools)
    install_github('nmdp-bioinformatics/service-gfe-submission/client-R')
}
library('gfeClient')

host <- 'http://gfe.b12x.org/'

# Get GFE from fasta file
fasta.file <- 'GFE_Submission/t/resources/fastatest1.fasta'
fasta.gfe  <- fasta2gfe(host,'HLA-A',fasta.file)

# Get sequence from
seq        <- gfe2seq(host,'HLA-A','HLA-Aw1-1-7-20-10-32-7-1-1-1-6-1-5-3-5-1-1')
```

```
# Get GFE from sequence
gfe         <- seq2gfe(host,'HLA-A',seq)

# return detailed logs
verbose     <- 1
gfe         <- seq2gfe(host,'HLA-A',seq,verbose)

# Return structure (ex. exon, 1 , TGCCCAAGCCCCTCACCCTGAGATGGG)
structure   <- 1
gfe         <- seq2gfe(host,'HLA-A',seq,verbose,structure)
```

3.3 Perl Client

The perl client is available in the *client-perl* directory in the github repositiory. Here are a few examples of installing and using the perl client:

```
#!/usr/bin/env perl
use strict;
use warnings;
use GFE_Client;

my $s_seq    = shift @ARGV;
my $s_locus = shift @ARGV;

# Does alignment of sequence and submission of aligned
# sequence to the GFE service.
my $o_client = GFE_Client->new();
my $rh_gfe    = $o_client->getGfe($s_locus,$s_seq);

# Print out GFE
print $$rh_gfe{gfe},"\n";
```

CHAPTER 4

Tools

Three tools are available in the client-perl directory in the GFE Service github repository. When running these tools you can specify which feature-service and gfe-service to use. Without specifying they default to feature.nmdp-bioinformatics.org and gfe.b12x.org.

4.1 Installing Tools

Note: Some perl modules will fail to load if you don't have certain software installed. Make sure you have *libssl-dev* installed otherwise *Moose* and *Log4j* will fail to properly install.

1. Clone the github repository.
2. Change to the client-perl directory with the *Makefile.PL*.
3. Install cpanm and all the perl dependencies. `curl -LO http://xrl.us/cpanm && perl cpanm --notest --installdeps .`
4. Run *make test* and *make install*. `perl Makefile.PL && make && make test && make install`

4.2 Tool Documentation

4.2.1 seq2gfe

Tip: Use seq2gfe for quickly investigating a particular sequence. Don't try to use it for bulk analysis of sequences.

Parameter	Description
-s/–seq	Raw sequence text, defaults to `STDIN`
-u/–url	URL for service, default is gfe.b12x.org
-l/–locus	Gene locus
-v/–verbose	Flag for running in verbose
-h/–help	Flag for returning the Perl POD

Example commands:

```
seq2gfe --seq GACGGCAAGGATTACATCGCCCTGAACGAGGACCTGCGCT \
CTTGGACCGCGGCGGACATGGCGGCTCAGATCACCAAGCGCAAGTACCTGCGCT -l HLA-A > seqtest1.gfe.csv
cat GACGGCAAGGATTACATCGCCCTGAACGAGGACCTGCGCTCTTGGACCGC \
GGCGGACATGGCGGCTCAGATCACCAAGCGCAAGTACCTGCGCTCTTGGACCGC | seq2gfe -l HLA-A > seqtest1.gfe.csv
```

4.2.2 fasta2gfe

Parameter	Description
-f/–fasta	Fasta file, defaults to `STDIN`
-u/–url	URL for service, default is gfe.b12x.org
-l/–locus	Gene locus
-v/–verbose	Flag for running in verbose
-h/–help	Flag for returning the Perl POD

Example commands:

```
fasta2gfe --fasta t/resources/fastatest1.fasta -l HLA-A > fastatest1.gfe.csv
cat t/resources/fastatest1.fasta | fasta2gfe -l HLA-A > fastatest1.gfe.csv
```

4.2.3 hml2gfe

Parameter	Description
-i/–input	Input HML File
-u/–url	URL for service, default is gfe.b12x.org
-m/–hml	Flag for returning results in HML file
-v/–verbose	Flag for running in verbose
-h/–help	Flag for returning the Perl POD

Example commands:

```
hml2gfe --input t/resources/hmltest1.HML > hmltest1.gfe.csv
hml2gfe --input t/resources/hmltest1.HML --hml > hmltest1.gfe.HML
```

CHAPTER 5

Docker

Note: Running and installing docker requires sudo access on the machine you're using.

This page walks through how to build the service-gfe-submission docker image and how to run the service from the docker image. To install docker, go to the Docker homepage and follow the installation process for the operating system you're using.

5.1 Building Docker Image Locally

Only build the docker image locally if you're making changes to the service and are trying to do some debugging.

```
cd service-gfe-submission/docker
docker build -t service-gfe-submssion:latest .
```

If the docker image is successfuly built then typing `docker images` will show a new image labeled service-gfe-submission. For running this image follow the instructions below, except remove the *nmdpbioinformatics* from the image name.

5.2 Pulling the Image

The easiest way to get the service running locally, is to pull an image containing the service from docker hub. Running the following command will pull the latest GFE service image from docker hub. The image on docker hub is built from the *Dockerfile* in the *docker* directory in the github repository. Every new commit to the *nmdp-bioinformatics/service-gfe-submission* repository triggers a new build of the docker image on docker hub.

Click here for more information on the publically available docker image.

Tip: If you want a particular verison of the GFE service, then you can specify what release version after the name in place of *latest*.

```
docker pull nmdpbioinformatics/service-gfe-submission
```

5.3 Running Service

Once the image is successfuly pulled to your machine you can run the service using the

```
docker run -d --name service-gfe-submission -p 8080:5050 nmdpbioinformatics/service-gfe-submission
```

The *-d* flag runs the service in "detached-mode" in the background and *-p* specifies what ports to expose. Make sure the ports you expose are not already in use. If the docker container is successfuly executed then typing `docker ps -a` will show a new container labeled *service-gfe-submission* running.

5.4 Debugging

Tip: The JSON parsing tool **jq** can be useful for parsing through the JSON docker logs.

If you want to stop and delete a currently running docker container, then run `docker ps` to find the container id and then run the following command.

```
docker kill {container id}
docker rm   {container id}
```

New releases of the GFE service will be available on docker hub in the coming months and can be pulled using the same command from above. Any new version you pull will have to be run with different ports or you will need to kill the container running the service. If you encounter an error while running the service that isn't

```
docker run --rm -it service-gfe-submission:latest /bin/bash
```

This allows you to enter the image through bash and access all of the files and programs installed on the image. If something is failing when you try to run the service, then try and run the service from within the image.

Refer to the docker documentation for more information on using docker.

CHAPTER 6

Installation

Note: The following instructions are for installing and running the service locally for **development purposes**. If you just want to run the service locally, then refer to the documentaiton on running the Docker image instead.

6.1 Software Requirements

Before installing the GFE service locally, you'll need to install some required software. If you want to reproduce the GFE service locally, then installing the *ngs-tools* and *annotation pipeline* from the file available in the docker directory is the safest method.

- Git
- Perl
- Java
- Annotation Pipeline
- nextflow
- ngs-tools

6.2 Perl Packages

Tip: Some perl modules will fail to load if you don't have certain software installed. Make sure you have *libssl-dev* installed otherwise *Moose* and *Log4j* will fail to properly install.

1. Clone the **'github repository'_**
2. Change into the *service-gfe-submission directory, with the *Makefile.PL* file.
3. Install cpanm and all the perl dependencies. `curl -LO http://xrl.us/cpanm && perl cpanm --notest --installdeps .`

6.3 ngs-tools

There are several ways that you can build the ngs-tools needed for running the GFE service. The easiest and safest way would be to install from the debian file located in the docker directory. Make sure to export the location the ngs-tools in your `PATH` environment variable.

6.3.1 Installing From Debian

If you're not worried about the mechanics of the ngs-tools, then installing it from the debian file located in the docker directory is the best method. Don't use this debian for the testing and development of the ngs-tools.

```
cd service-gfe-submission/docker
dpkg --install ngs-tools_1.9.deb
```

6.3.2 Building locally

If you want to make some modifications to the ngs-tools and test it locally with the GFE service, then this is the best option for you. If you think the changes you make should be part of the ngs code base, then make sure to make a pull request to the ngs master branch on github.

```
git clone https://github.com/nmdp-bioinformatics/ngs
cd ngs
mvn install
```

6.4 Annotation Pipeline

Currently the annotaion pipeline is not able to be built from the github page. Instead you can install it locally, by unzipping the tar file in the docker directory. We're currently working on adding this to the ngs code base on github.

```
cd service-gfe-submission/docker
tar -xzf hap1.2.tar.gz
export PATH=$PATH:`pwd`
```

6.5 Installing service-gfe-submission

Note: This step will fail if you don't have the required software listed above installed and listed in the `PATH` environment variable. Make sure to add the *nextflow*, *anootation pipeline* and the *ngs-tools* to your `PATH` environment variable.

1. Clone the github repository.
2. Change to the directory of the *Makefile.PL*.
3. Run *make test* and *make install*. `perl Makefile.PL && make && make test && make install`
4. Run the service locally. `plackup bin/app.pl`
5. Go to http://localhost:5000 for the GUI and access to the Swagger API.

6. Now test the service with `curl`.

```
curl --header "Content-type: application/json" --request POST \
--data '{"locus":"HLA-A","gfe":"HLA-Aw1-1-7-20-10-32-7-1-1-1-6-1-5-3-5-1-0"}' \
http://localhost:5000/sequence
```

CHAPTER 7

Contributing

1. Log into Github web interface with your username-nmdp account
2. Browse to the repo at https://github.com/nmdp-bioinformatics/flow-blast-hml, hit the Fork button.
3. Copy the clone URL from the Github web page for the fork (something like https://github.com/username-nmdp/pipeline.git)
4. Clone the fork

```
git clone https://github.com/username-nmdp/flow-blast-hml.git
cd flow-blast-hml
```

5. Add upstream as remote

```
git remote add upstream https://github.com/nmdp-bioinformatics/flow-blast-hml
```

6. Pull and merge latest changes from upstream master to your local master branch

```
git checkout master
git pull upstream master
git push
```

7. Create a new local feature branch

```
git checkout -b new-feature-branch
```

8. Edit files locally
9. Commit changes to local feature branch

```
git commit -m "made changes"
```

10. Push changes from local feature branch to remote feature branch on your fork

```
git push origin new-feature-branch
```

11. Browse to the Github web page for your fork repo (something like https://github.com/username-nmdp/flow-blast-hml) and hit the new pull request button.
12. Edit the pull request description and hit create new pull request button.
13. Other contributors will review the changes in the pull request.
14. When the pull request looks good, it will be merged into the master branch.
15. Hit the delete branch button to delete your remote feature branch (the commits have been merge upstream, so it is no longer necessary).

16. Delete your local feature branch

```
git branch -d new-feature-branch
```