14: Biological Sequence Data Model

Last Update: February 8, 2013.

Overview

The overview for this chapter consists of the following topics:

- Introduction
- Chapter Outline

Introduction

This chapter describes the NCBI Biological Sequence Data Model, with emphasis on the ASN.1 files and C++ API. ASN.1 type names and the corresponding C++ class or data member names are used almost interchangeably throughout the chapter. Another good source of information about the NCBI data model is:

Bioinformatics
A Practical Guide to the Analysis of Genes and Proteins
Second Edition (2001)
Edited by Andreas D. Baxevanis, B. F. Francis Ouellette
ISBN 0-471-38391-0

Chapter 2 - The NCBI Data Model

Chapter Outline

The following is an outline of the topics presented in this chapter:

Data Model

The Data Model section outlines the NCBI model for biotechnology information, which is centered on the concept of a biological sequence as a simple, linear coordinate system.

- Introduction
- Biological Sequences

Introduction

The NCBI sequence databases and software tools are designed around a particular model of biological sequence data. It is designed to provide a few unifying concepts which cross a wide range of domains, providing a path between the domains. Specialized objects are defined which are appropriate within a domain. In the following sections we will present the unifying ideas, then examine each area of the model in more detail.

Since we expect that computer technologies will continue to develop at a rapid rate, NCBI has made considerable investment of time and energy to ensure that our data and software tools are not too tightly bound to any particular computer platform or database technology. However, we also wish to embrace the intellectual rigor imposed by describing our data within a formal system and in a machine readable and checkable way. For this reason we have chosen to describe our data in Abstract Syntax Notation 1 (ASN.1; ISO 8824, 8825). Enough explanation will be given here to allow the reader to examine the data definitions. A much fuller description of ASN.1 and the NCBI software tools which use it appears in later chapters.

The data specification sections are arranged by ASN.1 module with detailed discussions of data objects defined in each and the software functions available to operate on those objects. Each ASN.1 defined object has a matching C++ language class. Each C++ class has at a minimum, functions to: create it, write it to an ASN.1 stream, read it from an ASN.1 stream, and destroy it. Many objects have additional functions. Some of these are described in the section on the module and some with more extensive interfaces are described in additional sections. Each module section begins with a description of the elements, followed by the full ASN.1 definition of the module. The C++ API is referenced with links.

This section provides an overview of all modules. Selected ASN.1 definitions are inserted into the body of the text as necessary. They are also described in the section on the appropriate module.

There are two major areas for which data objects have been defined. One is bibliographic data. It is clear that this class of information is central to all scientific fields within and outside of molecular biology so we expect these definitions to be widely useful. We have followed the American National Standard for Bibliographic References (ANSI Z39.29-1977) and consulted with the US Patent Office and professional librarians to ensure complete and accurate representation of citation information. Unlike biological data, this data is relatively well understood, so we hope that the bibliographic specification can be quite complete and stable. Despite its importance, the bibliographic specification will not be discussed further here, since it does not present ideas which may be novel to the reader.

The other major area of the specification is biological sequence data and its associated information. Here the data model attempts to achieve a number of goals. Biomedical information is a vast interconnected web of data which crosses many domains of discourse with very different ways of viewing the world. Biological science is very much like the parable of the blind men and elephant. To some of the blind men the elephant feels like a column, to some like a snake, to others like a wall. The excitement of modern biological research is that we all agree that, at least at some level, we are all exploring aspects of the same thing. But it is early enough in the development of the science that we cannot agree on what that thing is.

The power of molecular biology is that DNA and protein sequence data cut across most fields of biology from evolution to development, from enzymology to agriculture, from statistical mechanics to medicine. Sequence data can be viewed as a simple, relatively well defined armature on which data from various disciplines can be hung. By associating diverse data with the sequence, connections can be made between fields of research with no other common ground, and often with little or no idea of what the other field is doing.

This data model establishes a biological sequence as a simple integer coordinate system with which diverse data can be associated. It is reasonable to hope that such a simple core can be very stable and compatible with a very wide range of data. Additional information closely linked to the coordinate system, such as the sequence of amino acids or bases, or genes on a genetic map are layered onto it. With stable identifiers for specific coordinate systems, a greater diversity of information about the coordinate system can be specifically attached to it in a very flexible yet rigorous way. The essential differences between different biological forms are preserved, yet they can viewed as aspects of the same thing around the core, and thus move us toward our goal of understanding the totality.

Biological Sequences

A Bioseq is a single continuous biological sequence. It can be nucleic acid or protein. It can be fully instantiated (i.e. we have data for every residue) or only partially instantiated (e.g. we know a fragment is 10 kilobases long, but we only have sequence data over 1 kilobase). A Bioseq is defined in ASN.1 as follows:

```
Bioseq ::= SEQUENCE {
 id SET OF Seq-id , -- equivalent identifiers
 descr Seq-descr OPTIONAL , -- descriptors
 inst Seq-inst , -- the sequence data
 annot SET OF Seq-annot OPTIONAL }
```

In ASN.1 a named datatype begins with a capital letter (e.g. Bioseq). The symbol "::=" means "is defined as". A primitive type is all capitals (e.g. SEQUENCE). A field within a named datatype begins with a lower case letter (e.g. descr). A structured datatype is bounded by curly brackets ({}). We can now read the definition above: a Bioseq is defined as a SEQUENCE (i.e. a structure where the elements must come in order; the mathematical notion of a sequence, not the biological one). The first element of Bioseq is called "id" and is a SET OF (i.e. an unordered collection of elements of the same type) a named datatype called "Seq-id". Seq-id would have its own definition elsewhere. The second element is called "descr" and is a named type called "Seq-descr", which is optional. In this text, when we wish to refer to the id element of the named type Bioseq, we will use the notation "Bioseq.id".

A Bioseq has two optional elements, which both have descriptive information about the sequence. Seq-descr is a collection of types of information about the context of the sequence. It may set biological context (e.g. define the organism sequenced), or bibliographic context (e.g. the paper it was published in), among other things. Seq-annot is information that is explicitly tied to locations on the sequence. This could be feature tables, alignments, or graphs, at the present time. A Bioseq can have more than one feature table, perhaps coming from different sources, or a feature table and a graph, etc.

A Bioseq is only required to have two elements, id and inst. Bioseq.id is a set of one or more identifiers for this Bioseq. An identifier is a key which allows us to retrieve this object from a database or identify it uniquely. It is not a name, which is a human compatible description, but not necessarily a unique identifier. The name "Jane Doe" does not uniquely identify a person in the United States, while the identifier, social security number, does. Each Seq-id is a CHOICE of one of a number of identifier types from different databases, which may have different structures. All Bioseqs *must* have at least one identifier.

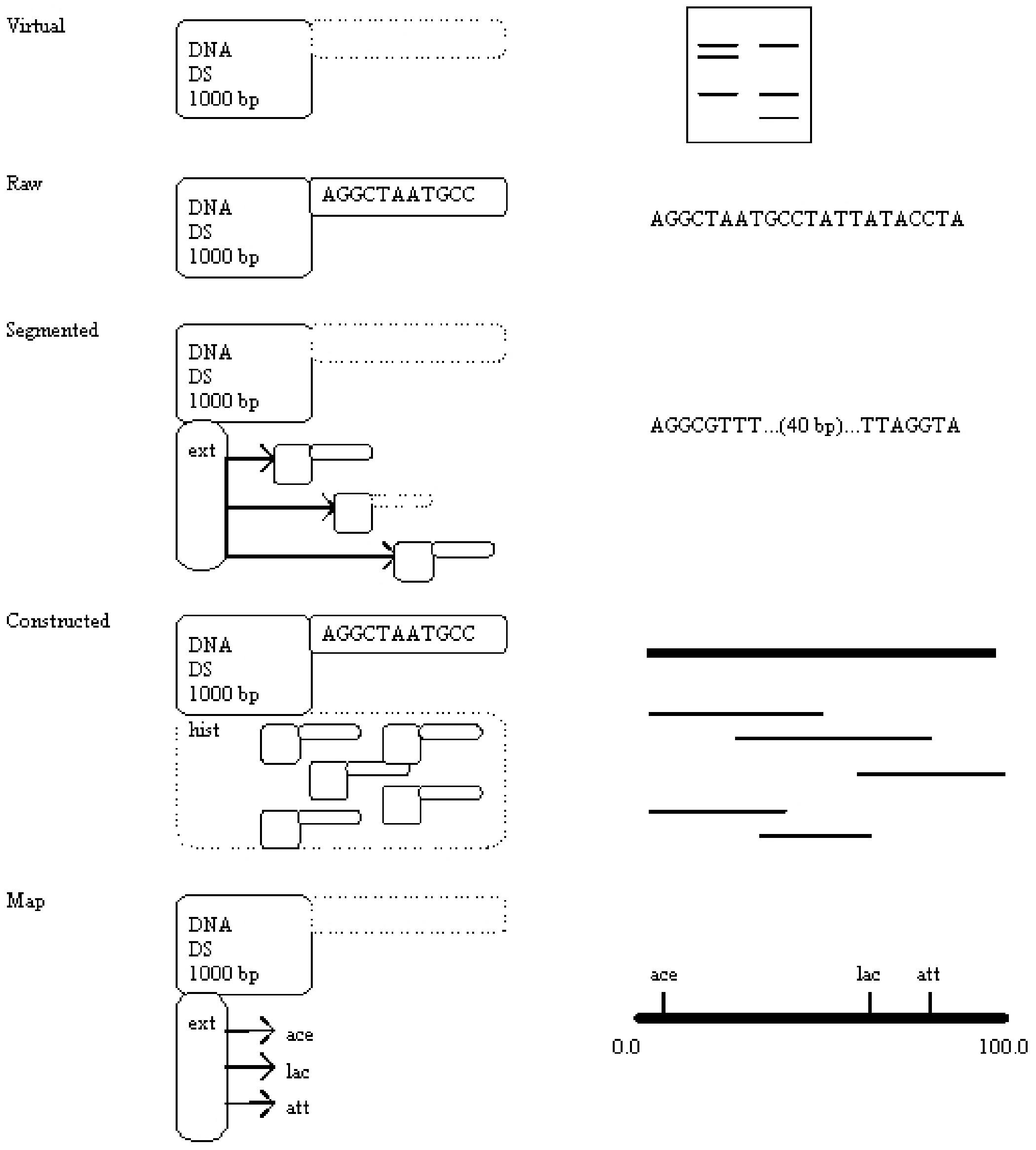
Classes of Biological Sequences

The other required element of a Bioseq is a Seq-inst. This element instantiates the sequence itself. It represents things like: Is it DNA, RNA, or protein? Is it circular or linear? Is it double-stranded or single-stranded? How long is it?

```
Seq-inst ::= SEQUENCE { -- the sequence data itself
 repr ENUMERATED { -- representation class
 not-set (0) , -- empty
 virtual (1) , -- no seq data
 raw (2) , -- continuous sequence
 seg (3) , -- segmented sequence
 const (4) , -- constructed sequence
 ref (5) , -- reference to another sequence
 consen (6) , -- consensus sequence or pattern
 map (7) , -- ordered map of any kind
 delta (8) , -- sequence made by changes (delta) to others
 other (255) } ,
 mol ENUMERATED { -- molecule class in living organism
 not-set (0) , -- > cdna = rna
 dna (1) ,
 rna (2) ,
 aa (3) ,
 na (4) , -- just a nucleic acid
 other (255) } ,
 length INTEGER OPTIONAL , -- length of sequence in residues
 fuzz Int-fuzz OPTIONAL , -- length uncertainty
 topology ENUMERATED { -- topology of molecule
 not-set (0) ,
 linear (1) ,
 circular (2) ,
 tandem (3) , -- some part of tandem repeat
 other (255) } DEFAULT linear ,
 strand ENUMERATED { -- strandedness in living organism
 not-set (0) ,
 ss (1) , -- single strand
 ds (2) , -- double strand
 mixed (3) ,
 other (255) } OPTIONAL , -- default ds for DNA, ss for RNA, pept
 seq-data Seq-data OPTIONAL , -- the sequence
 ext Seq-ext OPTIONAL , -- extensions for special types
 hist Seq-hist OPTIONAL } -- sequence history
```

Seq-inst is the parent class of a sequence representation class hierarchy. There are two major branches to the hierarchy. The molecule type branch is indicated by Seq-inst.mol. This could be a nucleic acid, or further sub classified as RNA or DNA. The nucleic acid may be circular, linear, or one repeat of a tandem repeat structure. It can be double, single, or of a mixed strandedness. It could also be a protein, in which case topology and strandedness are not relevant.

There is also a representation branch, which is independent of the molecule type branch. This class hierarchy involves the particular data structure used to represent the knowledge we have about the molecule, no matter which part of the molecule type branch it may be in. The repr element indicates the type of representation used. The aim of such a set of representation classes is to support the information to express different views of sequence based objects, from chromosomes to restriction fragments, from genetic maps to proteins, within a single overall model. The ability to do this confers profound advantages for software tools, data storage and retrieval, and traversal of related sequence and map data from different scientific domains.



A **virtual** representation is used to describe a sequence about which we may know things like it is DNA, it is double stranded, we may even know its length, but we do not have the actual sequence itself yet. Most fields of the Seq-inst are filled in, but Seq-inst.seq-data is empty. An example would be a band on a restriction map.

A **raw** representation is used for what we traditionally consider a sequence. We know it is DNA, it is double stranded, we know its length exactly, and we have the sequence data itself. In this case, Seq-inst.seq-data contains the sequence data.

A **segmented** representation is very analogous to a virtual representation. We posit that a continuous double stranded DNA sequence of a certain length exists, and pieces of it exist in other Bioseqs, but there is no data in Seq-inst.seq-data. Such a case would be when we have cloned and mapped a DNA fragment containing a large protein coding region, but have only actually sequenced the regions immediately around the exons. The sequence of each exon is an individual raw Bioseq in its own right. The regions between exons are virtual Bioseqs. The segmented Bioseq uses Seq-inst.ext to hold a SEQUENCE OF Seq-loc. That is, the extension is an ordered series of locations on *other* Bioseqs, in this case the raw and virtual Bioseqs representing the exons and introns. The segmented Bioseq contains data only by reference to other Bioseqs. In order to retrieve the base at the first position in the segmented Bioseq, one would go to the first Seq-loc in the extension, and return the appropriate base from the Bioseq it points to.

A **constructed** Bioseq is used to describe an assembly or merge of other Bioseqs. It is analogous to the raw representation. In fact, most raw Bioseqs were actually constructed from an assembly of gel readings. However, the constructed representation class is really meant for tracking higher level merging, such as when an expert in a particular organism or gene region may construct a "typical" sequence from that region by merging available sequence data, often published by different groups, using domain knowledge to resolve discrepancies between reports or to select a typical allele. Seq-inst contains an optional Seq-hist object. Seq-hist contains a field called "assembly" which is a SET OF Seq-align, or sequence alignments. The alignments are used to record the history of how the various component Bioseqs used for the merge are related to the final product. A constructed sequence DOES contain sequence data in Seq-inst.seq-data, unlike a segmented sequence, because the component sequences may overlap, or expert knowledge may have been used to determine the "correct" residue at any position that is not captured in the original components. So Seq-hist.assembly is used to simply record the relationship of the merge to the old Bioseqs, but does NOT describe how to generate it from them.

A **map** is akin to a virtual Bioseq. For example, for a genetic map of E.coli, we might posit that the E.coli chromosome is about 5 million base pairs long, DNA, double stranded, circular, but we do not have the sequence data for it. However, we do know the positions of some genes on this putative sequence. In this case, the Seq-inst.ext is a SEQUENCE OF Seq-feat, that is, a feature table. For a genetic map, the feature table contains Gene-ref features. An ordered restriction map would have a feature table containing Rsite-ref features. The feature table is part of Seq-inst because, for a map, it is an essential part of instantiating the map Bioseq, not merely annotation on a known sequence. In a sense, for a map, the annotation IS part of the sequence. As an aside, note that we have given gene positions on the E.coli genetic map in base pairs, while the standard E.coli map is numbered from 0.0 to 100.0 map units. Numbering systems can be applied to a Bioseq as a descriptor or a feature. For E.coli, we would simply apply the 0.0 - 100.0 floating point numbering system to the map Bioseq. Gene positions can then be shown to the scientists in familiar map units, while the underlying software still treats positions as large integers, just the same as with any other Bioseq.

Coordinates on ANY class of Bioseq are ALWAYS integer offsets. So the first residue in any Bioseq is at position 0. The last residue of any Bioseq is in position (length - 1).

The consequence of this design is that one uses EXACTLY the same data object to describe the location of a gene on an unsequenced restriction fragment, a fully sequenced piece of DNA, a partially sequenced piece of DNA, a putative overview of a large genetic region, or a genetic or physical map. Software to display, manipulate, or compare gene locations can work without change on the full range of possible representations. Sequence and physical map data can be easily integrated into a single, dynamically assembled view by creating a segmented sequence which points alternatively to raw or constructed Bioseqs and parts of a map Bioseq. The relationship between a genetic and physical map is simply an alignment between two Bioseqs of representation class map, no different than the alignment between two sequences of class raw generated by a database search program like BLAST or FASTA.

Locations on Biological Sequences

A Seq-loc is an object which defines a location on a Bioseq. The smooth class hierarchy for Seq-inst makes it possible to use the same Seq-loc to describe an interval on a genetic map as that used to describe an interval on a sequenced molecule.

Seq-loc is itself a class hierarchy. A valid Seq-loc can be an interval, a point, a whole sequence, a series of intervals, and so on.

```
Seq-loc ::= CHOICE {
 null NULL , -- not placed
 empty Seq-id , -- to NULL one Seq-id in a collection
 whole Seq-id , -- whole sequence
 int Seq-interval , -- from to
 packed-int Packed-seqint ,
 pnt Seq-point ,
 packed-pnt Packed-seqpnt ,
 mix Seq-loc-mix ,
 equiv Seq-loc-equiv , -- equivalent sets of locations
 bond Seq-bond ,
 feat Feat-id } -- indirect, through a Seq-feat
```

Seq-loc.null indicates a region of unknown length for which no data exists. Such a location may be used in a segmented sequence for the region between two sequenced fragments about which nothing, not even length, is known.

All other Seq-loc types, except Seq-loc.feat, contain a Seq-id. This means they are independent of context. This means that data objects describing information ABOUT Bioseqs can be created and exchanged independently from the Bioseq itself. This encourages the development and exchange of structured knowledge about sequence data from many directions and is an essential goal of the data model.

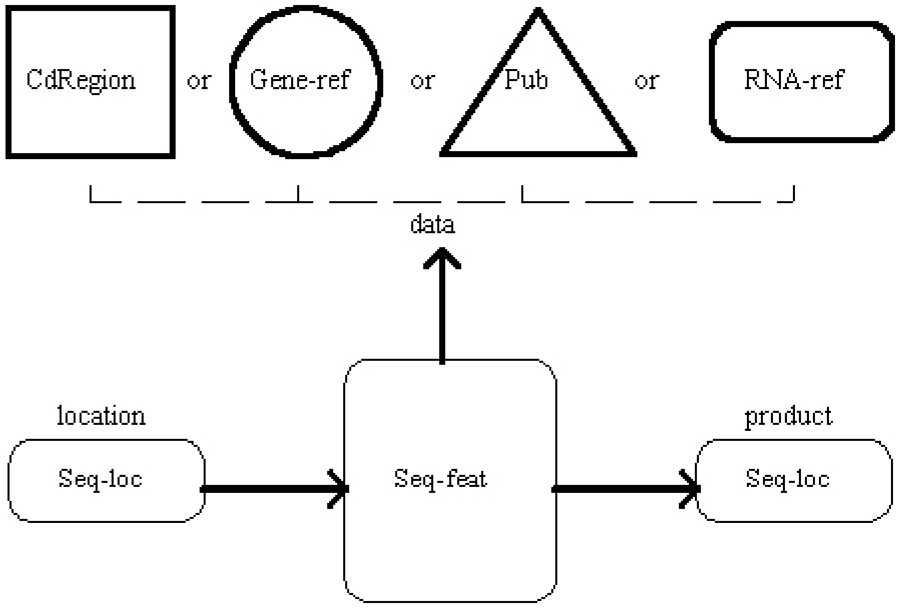
Associating Annotations With Locations On Biological Sequences

Seq-annot, or sequence annotation, is a collection of information ABOUT a sequence, tied to specific regions of Bioseqs through the use of Seq-loc's. A Bioseq can have many Seq-annot's associated with it. This allows knowledge from a variety of sources to be collected in a single place but still be attributed to the original sources. Currently there are three kinds of Seq-annot, feature tables, alignments, and graphs.

Feature Tables

A feature table is a collection of Seq-feat, or sequence features. A Seq-feat is designed to tie a Seq-loc together with a datablock, a block of specific data. Datablocks are defined objects themselves, many of which are objects used in their own right in some other context, such as publications (CPub_Base) or references to organisms (Org-ref) or genes (Gene-ref). Some datablocks, such as coding regions (CdRegion) make sense only in the context of a Seq-loc. However, since by design there is no intention that one datablock need to have anything in common with any other datablock, each can be tailored exactly to do a particular job. If a change or addition is required to one datablock, no others are affected. In those cases where a pre-existing object from another context is used as a datablock, any software that can use that object can now operate on the feature as well. For example, a piece of code to display a publication can operate on a publication from a bibliographic database or one used as a sequence feature with no change.

Since the Seq-feat data structure itself and the Seq-loc used to attach it to the sequence are common to all features, it is also possible to support a class of operations over all features without regard to the different types of datablocks attached to them. So a function to determine all features in a particular region of a Bioseq need not care what type of features they are.



A Seq-feat is bipolar in that it contains up to two Seq-loc's. Seq-feat.location indicates the "source" and is the location similar to the single location in common feature table implementations. Seq-feat.product is the "sink". A CdRegion feature would have its Seq-feat.location on the DNA and its Seq-feat.product on the protein sequence produced. Used this way it defines the process of translating a DNA sequence to a protein sequence. This establishes in an explicit way the important relationship between nucleic acid and protein sequence databases.

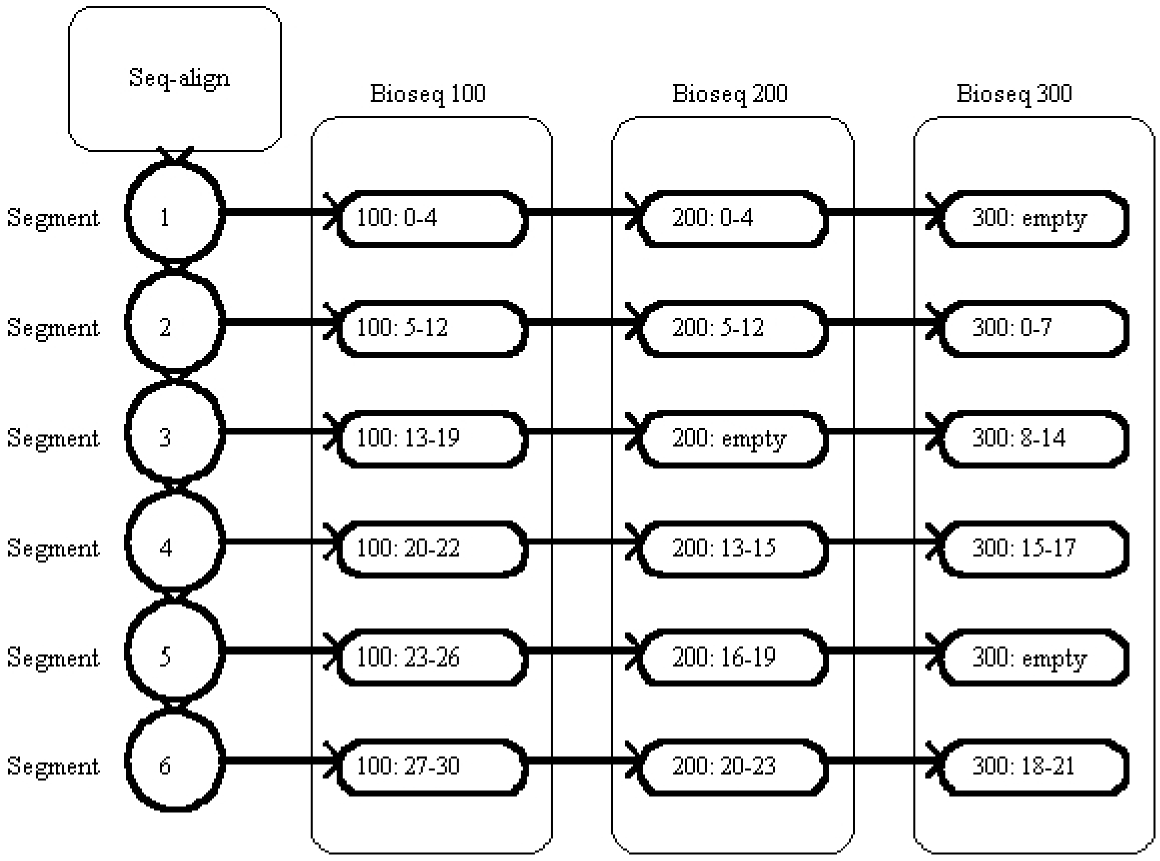
The presence of two Seq-loc's also allows a more complete representation of data conflicts or exceptional biological circumstances. If an author presents a DNA sequence and its protein product in a figure in a paper, it is possible to enter the DNA and protein sequences independently, then confirm through the CdRegion feature that the DNA in fact translates to that protein sequence. In an unfortunate number of published papers, the DNA presented does not translate to the protein presented. This may be a signal that the database has made an error of some sort, which can be caught early and corrected. Or the original paper may be in error. In this case, the "conflict" flag can be set in CdRegion, but the protein sequence is not lost, and retroactive work can be done to determine the source of the problem. It may also be the case that a genomic sequence cannot be translated to a protein for a known biological reason, such as RNA editing or suppressor tRNAs. In this case the "exception" flag can be set in Seq-feat to indicate that the data are correct, but will not behave in the expected way.

Sequence Alignments

A sequence alignment is essentially a correlation between Seq-locs, often associated with some score. An alignment is most commonly between two sequences, but it may be among many at once. In an alignment between two raw Bioseqs, a certain amount of optimization can be done in the data structure based on the knowledge that there is a one to one mapping between the residues of the sequences. So instead of recording the start and stop in Bioseq A and the start and stop in Bioseq B, it is enough to record the start in A and the start in B and the length of the aligned region. However if one is aligning a genetic map Bioseq with a physical map Bioseq, then one will wish to allow the aligned regions to distort relative one another to account for the differences from the different mapping techniques. To accommodate this most general case, there is a Seq-align type which is purely correlations between Seq-locs of any type, with no constraint that they cover exactly the same number of residues.

A Seq-align is considered to be a SEQUENCE OF segments. Each segment is an unbroken interval on a defined Bioseq, or a gap in that Bioseq. For example, let us look at the following three dimensional alignment with 6 segments:

```
Seq-ids
id=100 AAGGCCTTTTAGAGATGATGATGATGATGA
id=200 AAGGCCTaTTAG.......GATGATGATGA
id=300 ....CCTTTTAGAGATGATGAT....ATGA
| 1 | 2 | 3 | 4| 5 | 6 | Segments
```



The example above is a global alignment that is each segment sequentially maps a region of each Bioseq to a region of the others. An alignment can also be of type "diags", which is just a collection of segments with no implication about the logic of joining one segment to the next. This is equivalent to the diagonal lines that are shown on a dot-matrix plot.

The example above illustrates the most general form of a Seq-align, Std-seg, where each segment is purely a correlated set of Seq-loc. Two other forms of Seq-align allow denser packing of data for when only raw Bioseqs are aligned. These are Dense-seg, for global alignments, and Dense-diag for "diag" collections. The basic underlying model for these denser types is very similar to that shown above, but the data structure itself is somewhat different.

Sequence Graph

The third annotation type is a graph on a sequence, Seq-graph. It is basically a Seq-loc, over which to apply the graph, and a series of numbers representing values of the graph along the sequence. A software tool which calculates base composition or hydrophobic tendency might generate a Seq-graph. Additional fields in Seq-graph allow specification of axis labels, setting of ranges covered, compression of the data relative to the sequence, and so on.

Collections of Related Biological Sequences

It is often useful, even "natural", to package a group of sequences together. Some examples are a segmented Bioseq and the Bioseqs that make up its parts, a DNA sequence and its translated proteins, the separate chains of a multi-chain molecule, and so on. A Bioseq-set is such a collection of Bioseqs.

```
Bioseq-set ::= SEQUENCE { -- just a collection
 id Object-id OPTIONAL ,
 coll Dbtag OPTIONAL , -- to identify a collection
 level INTEGER OPTIONAL , -- nesting level
 class ENUMERATED {
 not-set (0) ,
 nuc-prot (1) , -- nuc acid and coded proteins
 segset (2) , -- segmented sequence + parts
 conset (3) , -- constructed sequence + parts
 parts (4) , -- parts for 2 or 3
 gibb (5) , -- geninfo backbone
 gi (6) , -- geninfo
 genbank (7) , -- converted genbank
 pir (8) , -- converted pir
 pub-set (9) , -- all the seqs from a single publication
 equiv (10) , -- a set of equivalent maps or seqs
 swissprot (11) , -- converted SWISSPROT
 pdb-entry (12) , -- a complete PDB entry
 mut-set (13) , -- set of mutations
 pop-set (14) , -- population study
 phy-set (15) , -- phylogenetic study
 eco-set (16) , -- ecological sample study
 gen-prod-set (17) , -- genomic products, chrom+mRNA+protein
 wgs-set (18) , -- whole genome shotgun project
 named-annot (19) , -- named annotation set
 named-annot-prod (20) , -- with instantiated mRNA+protein
 read-set (21) , -- set from a single read
 paired-end-reads (22) , -- paired sequences within a read-set
 other (255) } DEFAULT not-set ,
 release VisibleString OPTIONAL ,
 date Date OPTIONAL ,
 descr Seq-descr OPTIONAL ,
 seq-set SEQUENCE OF Seq-entry ,
 annot SET OF Seq-annot OPTIONAL }
```

The basic structure of a Bioseq-set is very similar to that of a Bioseq. Instead of Bioseq.id, there is a series of identifier and descriptive fields for the set. A Bioseq-set is only a convenient way of packaging sequences so controlled, stable identifiers are less important for them than they are for Bioseqs. After the first few fields the structure is exactly parallel to a Bioseq.

There are descriptors which describe aspects of the collection and the Bioseqs within the collection. The general rule for descriptors in a Bioseq-set is that they apply to "all of everything below". That is, a Bioseq-set of human sequences need have only one Org-ref descriptor for "human" at the top level of the set, and it is applied to all Bioseqs within the set.

Then follows the equivalent of Seq-inst, that is the instantiation of the data. In this case, the data is the chain of contained Bioseqs or Bioseq-sets. A Seq-entry is either a Bioseq or Bioseq-set. Seq-entry's are very often used as arguments to display and analysis functions, since one can move around either a single Bioseq or a collection of related Bioseqs in context just as easily. This also makes a Bioseq-set recursive. That is, it may consist of collections of collections.

```
Seq-entry ::= CHOICE {
 seq Bioseq ,
 set Bioseq-set }
```

Finally, a Bioseq-set may contain Seq-annot's. Generally one would put the Seq-annot's which apply to more than one Bioseq in the Bioseq-set at this level. Examples would be CdRegion features that point to DNA and protein Bioseqs, or Seq-align which align more than one Bioseq with each other. However, since Seq-annot's always explicitly cite a Seq-id, it does not matter, in terms of meaning, at what level they are put. This is in contrast to descriptors, where context does matter.

Consequences of the Data Model

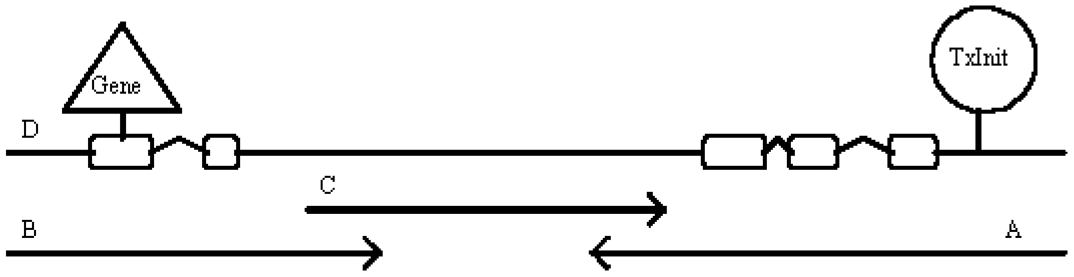
This data model has profound consequences for building sequence databases and for researchers and software tools interacting with them. Assuming that Seq-ids point to stable coordinate systems, it is easily possible to consider the whole set of data conforming to the model as a distributed, active heterogeneous database. For example, let us suppose that two raw Bioseqs with Seq-ids "A" and "B" are published in the scientific literature and appear in the large public sequence databases. They are both genomic nucleic acid sequences from human, each coding for a single protein.

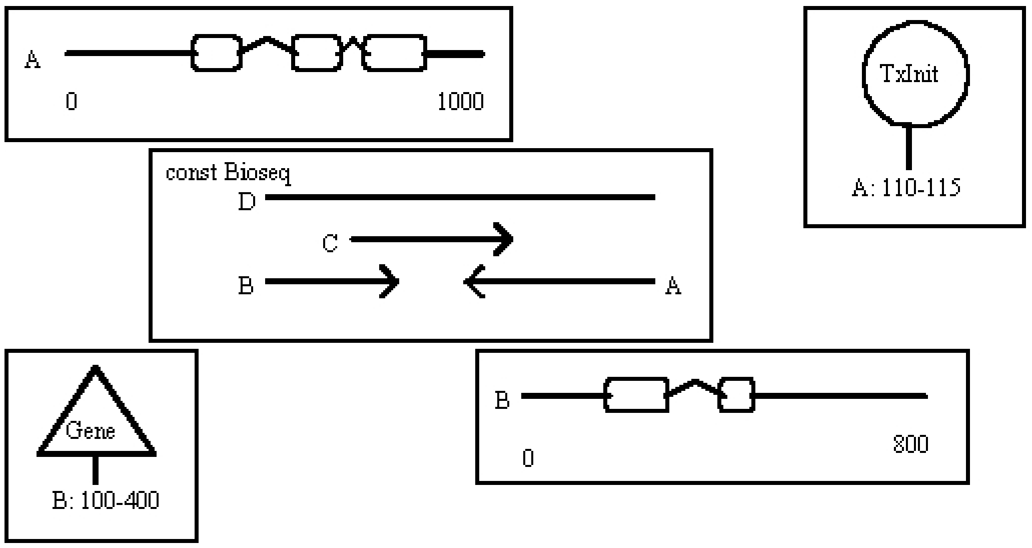
One researcher is a specialist in transcription initiation. He finds additional experimental information involving detailed work on initiation for the flanking region of Bioseq "A". He can then submit a feature table with a TxInit feature in it to the database with his summarized data. He need not contact the original author of "A", nor edit the original sequence entry for "A" to do this. The database staff, who are not experts in transcription initiation, need not attempt to annotate every transcription initiation paper in sufficient detail and accuracy to be of interest to a specialist in the area. The researcher submitting the feature need not use any particular software system or computer to participate, he need only submit a ASN.1 message which conforms to the specification for a feature.

Another researcher is a medical geneticist who is interested in the medical consequences of mutations in the gene on Bioseq "B". This individual can add annotation to "B" which is totally different in content to that added by the transcription specialist (in fact, it is unlikely that either follows the literature read by the other) and submit the data to the database in precisely the same way.

A third group may be doing bulk sequencing in the region of the human chromosome where "A" and "B" lie. They produce a third sequence, "C", which they discover by sequence similarity and mapping data, overlaps "A" at one end and "B" at the other. This group can submit not just the sequence of "C" but its relationship to "A" and "B" to the database and as part of their publication.

The database now has the information from five different research groups, experts in different fields, using different computer and software systems, and unaware, in many cases, of each other's work, to unambiguously pull together all this related information into an integrated high level view through the use of the shared data model and the controlled Seq-ids on common cited coordinate systems. This integration across disciplines and generation of high level views of the data is continuously and automatically available to all users and can be updated immediately on the arrival of new data without human intervention or interpretation by the database staff. This moves scientific databases from the role of curators of scientific data to the role of facilitators of discourse among researchers. It makes identification of potentially fruitful connections across disciplines an automatic result of data entry, rather than of painstaking analysis by a central group. It takes advantage of the growing rush of molecular biology data, making its volume and diversity advantages rather than liabilities.





Programming Considerations

To use the data model classes, add the following to your makefile:

```
LIB = general xser xutil xncbi
```

You will also need to include the relevant header files for the types you are using, and use the proper namespaces, for example:

```
#include <objects/general/Date_std.hpp>
USING_SCOPE(ncbi);
USING_SCOPE(ncbi::objects);
```

Types (such as Person-id) that contain other types can be constructed by assigning their contained types, beginning with the most nested level. For example, the following constructs a Person-id, which contains a Dbtag, which in turn contains an Object-id:

```
CObject_id obj;
obj.SetId(123);
```

```
CDbtag tag;
tag.SetDb("some db");
tag.SetTag(obj);

CPerson_id person;
person.SetDbtag(tag);
```

General Use Objects

This section describes the data objects defined in general.asn and their C++ classes and APIs. They are a miscellaneous collection of generally useful types.

- The Date: Date-std and Date
- Identifying Things: Object-id
- Identifying Things: Dbtag
- Identifying People: Name-std
- Identifying People: Person-id
- Expressing Uncertainty with Fuzzy Integers: Int-fuzz
- Creating Your Own Objects: User-object and User-field
- ASN.1 Specification: general.asn

The Date: Date-std and Date

ASN.1 has primitive types for recording dates, but they model a precise timestamp down to the minute, second, or even fraction of a second. For scientific and bibliographic data, it is common that only the date, or even just a portion of the date (e.g. month and year) is available - for example in a publication date. Rather than use artificial zero values for the unneeded fields of the ASN.1 types, we have created a specialized Date type. Date is a CHOICE of a simple, unparsed string or a structured Date-std. The string form is a fall-back for when the input data cannot be parsed into the standard date fields. It should only be used as a last resort to accommodate old data, as it is impossible to compute or index on.

When possible, the Date-std type should be used. In this case year is an integer (e.g. 1992), month is an integer from 1-12 (where January is 1), and day is an integer from 1-31. A string called "season" can be used, particularly for bibliographic citations (e.g. the "spring" issue). When a range of months is given for an issue (e.g. "JuneJuly") it cannot be represented directly. However, one would like to be able to index on integer months but still not lose the range. This is accomplished by putting 6 in the "month" slot and "July" in the "season" slot. Then the fields can be put back together for display and the issue can still be indexed by month. Year is the only required field in a Date-std.

The Date type can accommodate both the representation of the CHOICE itself (which kind of Date is this?) and the data for either CHOICE.

The Date and Date-std types are implemented with the CDate and CDate_std classes.

The CDate class should be used to create dates that can't be parsed into standard fields, for example:

```
CDate adate;
adate.SetStr("The birth of modern genetics.");
```

The CDate_std class should be used for parseable dates, i.e. dates with a given year, and optionally a month and day:

```
CDate_std adate;
adate.SetYear(2009);
adate.SetSeason("almost fall");
```

To include a time in the date:

```
CDate_std adate(CTime(CTime::eCurrent));
adate.SetSeason("late summer");
```

Identifying Things: Object-id

An Object-id is a simple structure used to identify a data object. It is just a CHOICE of an INTEGER or a VisibleString. It must always be used within some defining context (e.g. see Dbtag below) in order to have some global meaning. It allows flexibility in a host system's preference for identifying things by integers or strings.

The Object-id type is implemented by the CObject_id class. CObject_id includes the Match(), Compare(), and operator<() methods for determining whether two Object-id's are identical.

Types that include choices, such as Object-id, retain the last CHOICE assigned to them. For example, the following results in the Object-id being a string:

```
CObject_id obj;
obj.SetId(123);
obj.SetStr("some object");
```

Identifying Things: Dbtag

A Dbtag is an Object-id within the context of a database. The database is just defined by a VisibleString. The strings identifying the database are not centrally controlled, so it is possible that a conflict could occur. If there is a proliferation of Dbtags, then a registry might be considered at NCBI. Dbtags provide a simple, general way for small database providers to supply their own internal identifiers in a way which will, usually, be globally unique as well, yet requires no official sanction. So, for example, identifiers for features on sequences are not widely available at the present time. However, the Eukaryotic Promotor Database (EPD) can be provided as a set of features on sequences. The internal key to each EPD entry can be propagated as the Feature-id by using a Dbtag where "EPD" is the "db" field and an integer is used in the Object-id, which is the same integer identifying the entry in the normal EPD release.

The Dbtag type is implemented by the CDbtag class.

Identifying People: Name-std

A Name-std is a structured type for representing names with readily understood meanings for the fields. The full field is free-form and can include any or all of the other fields. The suffix field can be used for things like "Jr", "Sr", "III", etc. The title field can be used for things like "Dr.", "Sister", etc.

The Name-std type is implemented by the CName_std class.

Identifying People: Person-id

Person-id provides an extremely flexible way to identify people. There are five CHOICES from very explicit to completely unstructured. When one is building a database, one should select the most structured form possible. However, when one is processing data from other sources, one should pick the most structured form that adequately models the least structured input data expected.

The first Person-id CHOICE is a Dbtag. It would allow people to be identified by some formal registry. For example, in the USA, it might be possible to identify people by Social Security Number. Theoretically, one could then maintain a link to a person in database, even if they changed their name. Dbtag would allow other registries, such as professional societies, to be used as well. Frankly, this may be wishful thinking and possibly even socially inadvisable, though from a database standpoint, it would be very useful to have some stable identifier for people.

A Name-std CHOICE is the next most explicit form. It requires a last name and provides other optional name fields. This makes it possible to index by last name and disambiguate using one or more of the other fields (e.g. multiple people with the last name "Jones" might be distinguished by first name). This is the best choice when the data is available and its use should be encouraged by those building new databases wherever reasonable.

The next three choices contain just a single string. MEDLINE stores names in strings in a structured way (e.g. Jones JM). This means one can usually, but not always, parse out last names and can generally build indexes on the assumption that the last name is first. A consortium name can be used if the entity is a consortium rather than an individual, and finally a pure, unstructured string can be used.

The pure string form should be the CHOICE of last resort because no assumptions of any kind can be made about the structure of the name. It could be last name first, first name first, comma after last name, periods between initials, etc.

The Person-id type is implemented by the CPerson_id class.

Expressing Uncertainty with Fuzzy Integers: Int-fuzz

Lengths of Biological Sequences and locations on them are expressed with integers. However, sometimes it is desirable to be able to indicate some uncertainty about that length or location. Unfortunately, most software cannot make good use of such uncertainties, though in most cases this is fine. In order to provide both a simple, single integer view, as well as a more complex fuzzy view when appropriate, we have adopted the following strategy. In the NCBI specifications, all lengths and locations are always given by simple integers. If information about fuzziness is appropriate, then an Int-fuzz is ADDED to the data. In this case, the simple integer can be considered a "best guess" of the length or location. Thus simple software can ignore fuzziness, while it is not lost to more sophisticated uses.

Fuzziness can take a variety of forms. It can be plus or minus some fixed value. It can be somewhere in a range of values. It can be plus or minus a percentage of the best guess value. It may also be certain boundary conditions (greater than the value, less than the value) or refer to the bond BETWEEN residues of the biological sequence (bond to the right of this residue, bond to the left of that residue).

The Int-fuzz type is implemented by the CInt_fuzz class.

Creating Your Own Objects: User-object and User-field

One of the strengths of ASN.1 is that it requires a formal specification of data down to very detailed levels. This enforces clear definitions of data which greatly facilitates exchange of information in useful ways between different databases, software tools, and scientific enterprises. The problem with this approach is that it makes it very difficult for end users to add their own objects to the specification or enhance objects already in the specification. Certainly custom modules can be added to accommodate specific groups needs, but the data from such custom modules cannot be exchanged or passed through tools which adhere only to the common specification.

We have defined an object called a User-object, which can represent any class of simple, structured, or tabular data in a completely structured way, but which can be defined in any way that meets a user's needs. The User-object itself has a "class" tag which is a string used like the "db" string in Dbtag, to set the context in which this User-object is meaningful. The "class" strings are not centrally controlled, so again it is possible to have a conflict, but unlikely unless activity in this area becomes very great. Within a "class" one can define an object "type" by either a string or an integer. Thus any particular endeavor can define a wide variety of different types for their own use. The combination of "class" and "type" identifies the object to databases and software that may understand and make use this particular User-object's structure and properties. Yet, the generic definition means software that does not understand the purpose or use of any User-object can still parse it, pass it though, or even print it out for a user to peruse.

The attributes of the User-object are contained in one or more User-fields. Each User-field has a field label, which is either a string or an integer. It may contain any kind of data: strings; real numbers; integers; arrays of anything; or even sub-fields or complete sub-objects. When arrays and repeating fields are supplied, the optional "num" attribute of the User-field is used to tell software how many elements to prepare to receive. Virtually any structured data type from the simplest to the most complex can be built up from these elements.

The User-object is provided in a number of places in the public ASN.1 specifications to allow users to add their own structured features to Feature-tables or their own custom extensions to existing features. This allows new ideas to be tried out publicly, and allows software tools to be written to accommodate them, without requiring consensus among scientists or constant revisions to specifications. Those new ideas which time and experience indicate have become important concepts in molecular biology can be "graduated" to real ASN.1 specifications in the public scheme. A large body of structured data would presumably already exist in User-objects of this type, and these could all be back fitted into the new specified type, allowing data to "catch up" to the present specification. Those User-objects which do not turn out to be generally useful or important remain as harmless historical artifacts. User-objects could also be used for custom software to attach data only required for use by a particular tool to an existing standard object without harming it for use by standard tools.

The User-object and User-field types are implemented with the CUser_object and CUser_field classes.

Bibliographic References

The Bibliographic References section documents types for storing publications of any sort and collections of publications. The types are defined in biblio.asn and pub.asn modules.

Content

Introduction

The published literature is an essential component of any scientific endeavor, not just in molecular biology. The bibliographic component of the specification and the tools which go with it may find wide use then, permitting reuse of software and databases in many contexts. In addition, the fact that bibliographic citations appear in data from many sources, makes this data extremely valuable in linking data items from different databases to each other (i.e. indirectly through a shared literature citation) to build integrated views of complex data. For this reason, it is also important that database builders ensure that their literature component contain sufficient information to permit this mapping. By conforming to the specification below one can be assured that this will be the case.

Much of the following bibliographic specification was derived from the components recommended in the American National Standard for Bibliographic References (ANSI Z39.29-1977), and in interviews with professional librarians at the National Library of Medicine. The recommendations were then relaxed somewhat (by making certain fields OPTIONAL) to accommodate the less complete citation information available in current biomedical databases. Thus, although a field may be OPTIONAL, a database builder should still attempt to fill it, if it can reasonably be done.

In this section we also present a specification for the Pub type, publications of any sort and collections of publications. The MEDLINE specification has enough unique components that it is discussed separately in another section.

Citation Components: Affiliation

Affiliation is effectively the institutional affiliation of an author. Since it has the same fields needed to cite a publisher (of a book) it is reused in that context as well, although in that case it is not precisely an "affiliation". The Affil type is a CHOICE of two forms, a structured form which is preferred, or an unstructured string when that is all that is available.

The structured form has a number of fields taken from the ANSI guidelines. "affil" is institutional affiliation, such as "Harvard University". "div" is division within institution, such as "Department of Molecular Biology". "sub" is a subdivision of a country - in the United States this would be the state. "street" has been added to the specification (it is not included in ANSI) so that it is possible to produce a valid mailing address.

The Affil type is implemented by the CAffil class.

Citation Components: Authors

The Auth-list type represents the list of authors for the citation. It is a SEQUENCE, not a SET, since the order of author names matters. The names can be unstructured strings (the least desirable), semi-structured strings following the MEDLINE rules (e.g. "Jones JM"), or fully structured Author type objects (most desirable). An Affil can be associated with the whole list (typical of a scientific article). A more detailed discussion on the use of different types of names can be found in the "Identifying People" section of the "General Use Objects" section.

If fully structured Authors are used, each Author can have an individual Affil. The Author uses Person-id as defined above. The structured form also allows specification of the role of individual authors in producing the citation. The primary author(s) does not mean the "first" author, but rather that this author had a role in the original writing or experimental work. A secondary author is a reviewer or editor of the article. It is rare in a scientific work that a secondary author is ever mentioned by name. Authors may play different roles in the work, compiling, editing, and translating. Again, in a scientific work, the authors mentioned did none of these things, but were involved in the actual writing of the paper, although it would not be unusual anymore for one author to be the patent assignee. For scientific work, then, the main advantages of using the Author form are the use of fielded names and of individual Affils. For a book, being able to indicate the editors vs. the authors is useful also.

The Auth-list type is implemented by the CAuth_list class and the Author type is implemented by the CAuthor class.

Citation Components: Imprint

Imprint provides information about the physical form in which the citation appeared, such as what volume and issue of a journal it was in. For the "date" a structured Date is preferred. While "volume", "issue", and "pages" are commonly integers, there are many cases where they are not pure integers (e.g. pages xvi-xvii or issue 10A). Pages is given as a single string to simplify input from different sources. The convention is first page (hyphen) last page, or just page if it is on a single page. "section" may be relevant to a book or proceedings. "pub" is an Affil used to give the publisher of a book. The Affil.affil field is used to give the name of the publisher. "cprt" is the copyright date for a book. "part-sup" is for part or supplement and is not part of ANSI, but is used by MEDLINE. "language" is for the original language of the publication, which is also used by MEDLINE, but is not part of the ANSI standard. "prepub" is not part of the ANSI standard, but was added by NCBI to accommodate citations for as yet unpublished papers that can accompany data directly submitted by authors to the database.

The Imprint type is implemented by the CImprint class.

Citation Components: Title

A published work may have a number of Titles, each playing a particular role in specifying the work. There is the title of a paper, the title of a book it appears in, or the title of the journal, in which case it may come from a controlled list of serials. There may also be an original title and a translated title. For these reasons, Title is a defined entity rather than just a string, to allow the roles to be specified explicitly. Certain types of Title are legal for an Article, but not for a Journal or a Book. Rather than make three overlapping definitions, one for Article Titles, one for Journal Titles, and one for Book Titles, we have made one Title type and just indicated in the comments of the specification whether a particular form of Title is legal for an Article, Journal, or Book. Title is a SET OF because a work may have more than one title (e.g. an original and a translated title, or an ISO journal title abbreviation and an ISSN).

Title can be of a number of types. "name" is the full title of an article, or the full name of a book or journal. "tsub" is a subordinate title (e.g. "Hemoglobin Binds Oxygen" might be a primary title, while "Heme Groups in Biology: Part II" might be a subordinate title). "trans" is the translated title. So for an English language database like MEDLINE which contains an article originally published in French, the French title is "name" and the English version of it is "trans".

"jta" is a journal title abbreviation. It is only valid for a journal name, obviously. "jta" does not specify what kind of abbreviation it is, so it is the least useful of the journal designations available and should only be used as a last resort. "iso-jta" is an International Standards Organization (ISO) journal title abbreviation. This is the preferred form. A list of valid iso-jta's is available from NCBI or the National Library of Medicine. "ml-jta" is a MEDLINE journal title abbreviation. MEDLINE pre-dates the ISO effort, so it does not use iso-jta's. "coden" is a six letter code for journals which is used by a number of groups, particularly in Europe. "issn" is a code used by publishers to identify journals. To facilitate the use of controlled vocabularies for journal titles, NCBI maintains a file of mappings between "name", "iso-jta", "ml-jta", "coden", and "issn" where it is possible, and this file is available upon request.

"abr" is strictly the abbreviated title of a book. "isbn" is similar to "issn" in that it is a publishers abbreviation for a book. "isbn" is very useful, but one must be careful since it is used by publishers to list books, and to a publisher a hard cover book is different from a paperback (they have different "isbn"s) even if they have the same title.

The Title type is implemented by the CTitle class.

Citing an Article

An article always occurs within some other published medium. It can be an article in a journal or a chapter or section in a book or proceedings. Thus there are two components to an article citation; a citation for the work it was published in and a citation for the article within that work. Cit-art.title is the Title of the article and Cit-art.authors are the authors of the article. The "from" field is used to indicate the medium the article was published in, and reuses the standard definitions for citing a journal, book, or proceedings.

The Cit-art type is implemented by the CCit_art class.

Citing a Journal

Cit-jour is used to cite an issue of a journal, not an article within a journal (see Cit-art, above). Cit-jour.title is the title of the journal, and Cit-jour.imp gives the date, volume, issue of the journal. Cit-jour.imp also gives the pages of an article within the issue when used as part of a

Cit-art. This is not the purest possible split between article and journal, book, or proceedings, but does have the practical advantage of putting all such physical medium information together in a single common data structure. A controlled list of journal titles is maintained by NCBI, and database builders are encouraged to use this list to facilitate exchange and linking of data between databases.

The Cit-jour type is implemented by the CCit_jour class.

Citing a Book

Cit-book is used to cite a whole book, not an article within a book (see Cit-art, above). Cit-book.title is the title of this particular book. Cit-book.coll is used if the book if part of a collection, or muti-volume set (e.g. "The Complete Works of Charles Darwin"). Cit-book.authors is for the authors or editors of the book itself (not necessarily of any particular chapter). Cit-book.imp contains the publication information about the book. As with a Cit-art, if the Cit-book is being used to cite a chapter in a book, the pages in given in Cit-book.imp.

The Cit-book type is implemented by the CCit_book class.

Citing a Proceedings

A proceedings is a book published as a result or byproduct of a meeting. As such it contains all the same fields as a Cit-book and an additional block of information describing the meeting. These extra fields are the meeting number (as a string to accommodate things like "10A"), the date the meeting occurred, and an OPTIONAL Affil to record the place of the meeting. The name of the organization or meeting is normally the book title. Don't be confused by things like the Proceedings of the National Academy of Sciences, USA, which is really a journal.

The Cit-proc type is implemented by the CCit_proc class.

Citing a Letter, Manuscript, or Thesis

A letter, manuscript, or a thesis share most components and so are grouped together under type Cit-let. They all require most of the attributes of a book, and thus Cit-let incorporates the Cit-book structure. Unlike a normal book, they will not have a copyright date. A letter or manuscript will not have a publisher, although a thesis may. In addition, a manuscript may have a manuscript identifier (e.g. "Technical Report X1134").

The Cit-let type is implemented by the CCit_let class.

Citing Directly Submitted Data

The Cit-sub type is used to cite the submission of data directly to a database, independent of any publication(s) which may be associated with the data as well. Authors (of the submission) and Date (in an Imprint) are required. The Affiliation of the Authors should be filled in the Author-list. Optionally one may also record the medium in which the submission was made.

The Cit-sub type is implemented by the CCit_sub class.

Citing a Patent

A full patent citation, Cit-pat conveys not only enough information to identify a patent (see below) but to characterize it somewhat as well. A patent has a title and authors, the country in which the patent was issued, a document type and number, and the date the patent was issued. Patents are grouped into classes based on the patent subject, and this may be useful to know. In addition, when a patent is first filed it is issued an application number (different from the document number assigned to the issued patent). For tracking purposes, or issues of precedence, it is also helpful to know the application number and filing date.

The Cit-pat type is implemented by the CCit_pat class.

Identifying a Patent

When citing a patent, it may be sufficient to merely unambiguously identify it, on the assumption that more extensive information will be available from some other source, given the identifier. The Id-pat type contains fields only for the country in which the patent was applied for, or issued in, then a CHOICE of the patent document number (if issued) or the application number (if pending).

The CId-pat type is implemented by the CId_pat class.

Citing an Article or Book which is In Press

A number of the fields in Cit-art and Cit-book are OPTIONAL, not only to allow incorporation of older, incomplete databases, but also to allow partial information for works submitted, or in press. One simply fills in as many of the fields in Cit-art or Cit-book as possible. One must also set the "pre-pub" flag in Imprint to the appropriate status. That's it. Once the work is published, the remaining information is filled in and the "pre-pub" flag is removed. NOTE: this does NOT apply to work which is "unpublished" or "personal communication", or even "in preparation" because one knows nothing about where or when (or if) it will ever be published. One must use a Cit-gen for this (below).

Special Cases: Unpublished, Unparsed, or Unusual

A generic citation, Cit-gen, is used to hold anything not fitting into the more usual bibliographic entities described above. Cit-gen.cit is a string which can hold a unparsable citation (if you can parse it into a structured type, you should). Sometimes it is possible to parse some things but not everything. In this case, a number of fields, such as authors, journal, etc., which are similar to those in the structured types, can be populated as much as possible, and the remainder of the unparsed string can go in "cit".

Less standard citation types, such as a MEDLINE unique identifier, or the serial numbers used in the GenBank flatfile can be accommodated by Cit-gen. An unpublished citation normally has authors and date filled into the structured fields. Often a title is available as well (e.g. for a talk or for a manuscript in preparation). The string "unpublished" can then appear in the "cit" field.

Software developed to display or print a Cit-gen must be opportunistic about using whatever information is available. Obviously it is not possible to assume that all Cit-gens can be displayed in a uniform manner, but in practice at NCBI we have found they can generally be made fairly regular.

The Cit-gen type is implemented by the CCit_gen class.

Accommodating Any Publication Type

The Pub type is designed to accommodate a citation of any kind defined in the bibliographic specification, the MEDLINE specification, and more. It can also accommodate a collection of publications. It is very useful when one wishes to be able to associate a bibliographic reference in a very general way with a software tool or data item, yet still preserve the attributes specific for each class of citation. Pub is widely used for this purpose in the NCBI specifications.

The Pub type is implemented by the CPub class.

Grouping Different Forms of Citation for a Single Work

In some cases a database builder may wish to present more than one form of citation for the same bibliographic work. For example, in a sequence entry from the NCBI Backbone database, it is useful to provide the MEDLINE uid (for use as a link by other software tools), the Cit-art (for display to the user), and a Cit-gen containing the internal NCBI Backbone identifier for this publication as the string "pub_id = 188824" (for use in checking the database by in-house staff) for the same article. The Pub-equiv type provides this capability. It is a SET OF Pub. Each element in the SET is an equivalent citation for the same bibliographic work. Software can examine the SET and select the form most appropriate to the job at hand.

The Pub-equiv type is implemented by the CPub_equiv class.

Sets of Citations

One often needs to collect a set of citations together. Unlike the Pub-equiv type, the Pub-set type represents a set of citations for DIFFERENT bibliographic works. It is a CHOICE of types for a mixture of publication classes, or for a collection of the same publication class.

The Pub-set type is implemented by the CPub_set class.

MEDLINE Data

This section is an introduction to MEDLINE and the structure of a MEDLINE record. It describes types defined in the medline.asn module.

Module Types

Introduction

MEDLINE is the largest and oldest biomedical database in the world. It is built at the National Library of Medicine (NLM), a part of NIH. At this writing it contains over seven million citations from the scientific literature from over 3500 different journals. MEDLINE is a bibliographic database. It contains citation information (e.g. title, authors, journal, etc.). Many entries contain the abstract from the article. All articles are carefully indexed by professionals according to formal guidelines in a variety of ways. All entries can be uniquely identified by an integer key, the MEDLINE unique identifier (MEDLINE uid).

MEDLINE is a valuable resource in its own right. In addition, the MEDLINE uid can serve as a valuable link between entries in factual databases. When NCBI processes a new molecular biology factual database into the standardized format, we also normalize the bibliographic citations and attempt to map them to MEDLINE. For the biomedical databases we have tried thus far, we have succeeding in mapping most or all of the citations this way. From then on, linkage to other data objects can be made simply and easily through the shared MEDLINE uid. The MEDLINE uid also allows movement from the data item to the world of scientific literature in general and back.

Structure of a MEDLINE Entry

Each Medline-entry represents a single article from the scientific literature. The MEDLINE uid is an INTEGER which uniquely identifies the entry. If corrections are made to the contents of the entry, the uid is not changed. The MEDLINE uid is the simplest and most reliable way to identify the entry.

The entry-month (em) is the month and year in which the entry became part of the public view of MEDLINE. It is not the same as the date the article was published. It is mostly useful for tracking what is new since a previous query of MEDLINE.

The article citation itself is contained in a standard Cit-art, imported from the bibliographic module, so will not be discussed further here. The entry often contains the abstract from the article. The rest of the entry consists of various index terms, which will be discussed below.

MeSH Index Terms

Medical Subject Heading (MeSH) terms are a tree of controlled vocabulary maintained by the Library Operations division of NLM. The tree is arranged with parent terms above more specialized terms within the same concept. An entry in MEDLINE is indexed by the most specific MeSH term(s) available. Since the MeSH vocabulary is a tree, one may then query on specific terms directly, or on general terms by including all the child terms in the query as well.

A MeSH term may be qualified by one or more sub-headings. For example, the MeSH term "insulin" may carry quite a different meaning if qualified by "clinical trials" versus being qualified by "genetics".

A MeSH term or a sub-heading may be flagged as indicating the "main point" of the article. Again the most specific form is used. If the main point of the article was about insulin and they also discuss genetics, then the insulin MeSH term will be flagged but the genetics sub-heading will not be. However, if the main point of the article was the genetics of insulin, then the sub-heading genetics under the MeSH term insulin will be flagged but the MeSH term itself will not be.

Substance Records

If an article has substantial discussion of recognizable chemical compounds, they are indexed in the substance records. The record may contain only the name of the compound, or it may contain the name and a Chemical Abstracts Service (CAS) registry number or a Enzyme Commission (EC) number as appropriate.

Database Cross Reference Records

If an article cites an identifier recognized to be from a known list of biomedical databases, the cross reference is given in this field and the key for which database it was from. A typical example would be a GenBank accession number citing in an article.

Funding Identifiers

If an id number from a grant or contract is cited in the article (usually acknowledging support) it will appear in this field.

Gene Symbols

As an experiment, Library Operations at the NLM is putting in mnemonic symbols from articles, if they appear by form and usage to be gene symbols. Obviously such symbols vary and are not always properly used, so this field must be approached with caution. Nonetheless it can provide a route to a rich source of potentially relevant citations.

Biological Sequences

This section describes types used to represent biological data. These types are defined in the seq.asn, seqblock.asn, and seqcode.asn modules.

C++ Implementation Notes

Introduction

A biological sequence is a single, continuous molecule of nucleic acid or protein. It can be thought of as a multiple inheritance class hierarchy. One hierarchy is that of the underlying molecule type: DNA, RNA, or protein. The other hierarchy is the way the underlying biological sequence is represented by the data structure. It could be a physical or genetic map, an actual sequence of amino acids or nucleic acids, or some more complicated data structure building a composite view from other entries. An overview of this data model has been presented previously, in the Data Model section. The overview will not be repeated here so if you have not read that section, do so now. This section will concern itself with the details of the specification and representation of biological sequence data.

Bioseq: the Biological Sequence

A Bioseq represents a single, continuous molecule of nucleic acid or protein. It can be anything from a band on a gel to a complete chromosome. It can be a genetic or physical map. All Bioseqs have more common properties than differences. All Bioseqs must have at least one identifier, a Seq-id (i.e. Bioseqs must be citable). Seq-ids are discussed in detail in the Sequence Ids and Locations section. All Bioseqs represent an integer coordinate system (even maps). All positions on Bioseqs are given by offsets from the first residue, and thus fall in the range from zero to (length - 1). All Bioseqs may have specific descriptive data elements (descriptors) and/or annotations such as feature tables, alignments, or graphs associated with them.

The differences in Bioseqs arise primarily from the way they are instantiated (represented). Different data elements are required to represent a map than are required to represent a sequence of residues.

The C++ class for a Bioseq (CBioseq) has a list of Seq-id's, a Seq-descr, and a list of Seq-annot's, mapping quite directly from the ASN.1. However, since a Seq-inst is always required for a Bioseq, those fields have been incorporated into the Bioseq itself. Serialization is handled by CSerialObject from which CBioseq derives.

Related classes, such as CSeqdesc, provide enumerations for representing types of description, molecule types, and sequence encoding types used in the CBioseq class. Sequence encoding is discussed in more detail below.

The C++ Toolkit introduced some new methods for Bioseq's:

- CBioseq(CSeq_loc, string) - constructs a new delta sequence from the Seq-loc. The string argument may be used to specify local Seq-id text for the new Bioseq.
- GetParentEntry - returns Seq-entry containing the Bioseq.
- GetLabel - returns the Bioseq label.
- GetFirstId - returns the first element from the Bioseq's Id list or null.
- IsNa - true if the Bioseq is a nucleotide.
- IsAa - true if the Bioseq is a protein.

In addition, many utility functions for working with Bioseqs and sequence data are defined in the CSeqportUtil class.

Seq-id: Identifying the Bioseq

Every Bioseq MUST have at least one Seq-id, or sequence identifier. This means a Bioseq is always citable. You can refer to it by a label of some sort. This is a crucial property for different software tools or different scientists to be able to talk about the same thing. There is a wide range of Seq-ids and they are used in different ways. They are discussed in more detail in the Sequence Ids and Locations section.

Seq-annot: Annotating the Bioseq

A Seq-annot is a self-contained package of sequence annotations, or information that refers to specific locations on specific Bioseqs. Every Seq-annot can have an Object-id for local use by software, a Dbtag for globally identifying the source of the Seq-annot, and/or a name and description for display and use by a human. These describe the whole package of annotations and make it attributable to a source, independent of the source of the Bioseq.

A Seq-annot may contain a feature table, a set of sequence alignments, or a set of graphs of attributes along the sequence. These are described in detail in the Sequence Annotation section.

A Bioseq may have many Seq-annots. This means it is possible for one Bioseq to have feature tables from several different sources, or a feature table and set of alignments. A collection of sequences (see Sets Of Bioseqs) can have Seq-annots as well. Finally, a Seq-annot can stand alone, not directly attached to anything. This is because each element in the Seq-annot has specific references to locations on Bioseqs so the information is very explicitly associated with Bioseqs, not implicitly associated by attachment. This property makes possible the exchange of information about Bioseqs as naturally as the exchange of the Bioseqs themselves, be it among software tools or between scientists or as contributions to public databases.

Some of the important methods for the CSeq_annot class are:

- AddName() - adds or replaces annotation descriptor of type name.
- AddTitle(), SetTitle() - adds or replaces annotation descriptor of type title.
- AddComment() - adds annotation descriptor of type comment.
- SetCreateDate(), SetUpdateDate() - add or set annotation create/update time.
- AddUserObject() - add a user-object descriptor.

Seq-descr: Describing the Bioseq and Placing It In Context

A Seq-descr is meant to describe a Bioseq (or a set of Bioseqs) and place it in a biological and/or bibliographic context. Seq-descrs apply to the whole Bioseq. Some Seq-descr classes appear also as features, when used to describe a specific part of a Bioseq. But anything appearing at the Seq-descr level applies to the whole thing.

The C++ implementation of CSeq_descr uses a list of CSeqdesc objects, where each object contains a choice indicating what kind of CSeqdesc it is as well as the data representation of that choice. The CSeqdesc_Base header file lists the choice enumeration which are summarized in the following table. The Value column shows the numeric value of the choice.

Seqdesc Choice Variants

Value	**Name**	**Explanation**
0	e_not_set	choice not set
1	e_Mol_type	role of molecule in life
2	e_Modif	modifying keywords of mol-type
3	e_Method	protein sequencing method used
4	e_Name	a commonly used name (e.g. "SV40")
5	e_Title	a descriptive title or definition
6	e_Org	(single) organism from which mol comes
7	e_Comment	descriptive comment (may have many)
8	e_Num	a numbering system for whole Bioseq
9	e_Maploc	a map location from a mapping database
10	e_Pir	PIR specific data
11	e_Genbank	GenBank flatfile specific data
12	e_Pub	Publication citation and descriptive info from pub
13	e_Region	name of genome region (e.g. B-globin cluster)
14	e_User	user defined data object for any purpose
15	e_Sp	SWISSPROT specific data
16	e_Dbxref	cross reference to other databases
17	e_Embl	EMBL specific data

18	e_Create_date	date entry was created by source database
19	e_Update_date	date entry last updated by source database
20	e_Prf	PRF specific data
21	e_Pdb	PDB specific data
22	e_Het	heterogen: non-Bioseq atom/molecule
23	e_Source	source of materials, includes Org-ref
24	e_Molinfo	info on the molecule and techniques

mol-type: The Molecule Type

A Seq-descr.mol-type is of type GIBB-mol. It is derived from the molecule information used in the GenInfo BackBone database. It indicates the biological role of the Bioseq in life. It can be genomic (including organelle genomes). It can be a transcription product such as pre-mRNA, mRNA, rRNA, tRNA, snRNA (small nuclear RNA), or scRNA (small cytoplasmic RNA). All amino acid sequences are peptides. No distinction is made at this level about the level of processing of the peptide (but see Prot-ref in the Sequence Features section). The type other-genetic is provided for "other genetic material" such a B chromosomes or F factors that are not normal genomic material but are also not transcription products. The type genomic-mRNA is provided to describe sequences presented in figures in papers in which the author has combined genomic flanking sequence with cDNA sequence. Since such a figure often does not accurately reflect either the sequence of the mRNA or the sequence of genome, this practice should be discouraged.

modif: Modifying Our Assumptions About a Bioseq

A GIBB-mod began as a GenInfo BackBone component and was found to be of general utility. A GIBB-mod is meant to modify the assumptions one might make about a Bioseq. If a GIBB-mod is not present, it does not mean it does not apply, only that it is part of a reasonable assumption already. For example, a Bioseq with GIBB-mol = genomic would be assumed to be DNA, to be chromosomal, and to be partial (complete genome sequences are still rare). If GIBB-mod = mitochondrial and GIBB-mod = complete are both present in Seqdesc, then we know this is a complete mitochondrial genome. A Seqdesc contains a list of GIBB-mods.

The modifier concept permits a lot of flexibility. So a peptide with GIBB-mod = mitochondrial is a mitochondrial protein. There is no implication that it is from a mitochondrial gene, only that it functions in the mitochondrion. The assumption is that peptide sequences are complete, so GIBB-mod = complete is not necessary for most proteins, but GIBB-mod = partial is important information for some. A list of brief explanations of GIBB-mod values follows:

GIBB-mod

Value	Name	Explanation
0	eGIBB_mod_dna	molecule is DNA in life
1	eGIBB_mod_rna	molecule is RNA in life
2	eGIBB_mod_extrachrom	molecule is extrachromosomal
3	eGIBB_mod_plasmid	molecule is or is from a plasmid

4	eGIBB_mod_mitochondrial	molecule is from mitochondrion
5	eGIBB_mod_chloroplast	molecule is from chloroplast
6	eGIBB_mod_kinetoplast	molecule is from kinetoplast
7	eGIBB_mod_cyanelle	molecule is from cyanelle
8	eGIBB_mod_synthetic	molecule was synthesized artificially
9	eGIBB_mod_recombinant	molecule was formed by recombination
10	eGIBB_mod_partial	not a complete sequence for molecule
11	eGIBB_mod_complete	sequence covers complete molecule
12	eGIBB_mod_mutagen	molecule subjected to mutagenesis
13	eGIBB_mod_natmut	molecule is a naturally occurring mutant
14	eGIBB_mod_transposon	molecule is a transposon
15	eGIBB_mod_insertion_seq	molecule is an insertion sequence
16	eGIBB_mod_no_left	partial molecule is missing left end 5' end for nucleic acid, NH3 end for peptide
17	eGIBB_mod_no_right	partial molecule is missing right end 3' end for nucleic acid, COOH end for peptide
18	eGIBB_mod_macronuclear	molecule is from macronucleus
19	eGIBB_mod_proviral	molecule is an integrated provirus
20	eGIBB_mod_est	molecule is an expressed sequence tag
21	eGIBB_mod_sts	sequence tagged site
22	eGIBB_mod_survey	one pass survey sequence
23	eGIBB_mod_chromoplast	
24	eGIBB_mod_genemap	genetic map
25	eGIBB_mod_restmap	ordered restriction map
26	eGIBB_mod_physmap	physical map (not ordered restriction map)
255	eGIBB_mod_other	

method: Protein Sequencing Method

The method GetMethod() gives the method used to obtain a protein sequence. The values for a GIBB-method are stored in the object as enumerated values mapping directly from the ASN. 1 ENUMERATED type. They are:

GIBB-method

Value	**Name**	**Explanation**
1	eGIBB_method_concept_trans	conceptual translation
2	eGIBB_method_seq_pept	peptide itself was sequenced
3	eGIBB_method_both	conceptual translation with partial peptide sequencing

4	eGIBB_method_seq_pept_overlap	peptides sequenced, fragments ordered by overlap
5	eGIBB_method_seq_pept_homol	peptides sequenced, fragments ordered by homology
6	eGIBB_method_concept_trans_a	conceptual translation, provided by author of sequence

name: A Descriptive Name

A sequence name is very different from a sequence identifier. A Seq-id uniquely identifies a specific Bioseq. A Seq-id may be no more than an integer and will not necessarily convey any biological or descriptive information in itself. A name is not guaranteed to uniquely identify a single Bioseq, but if used with caution, can be a very useful tool to identify the best current entry for a biological entity. For example, we may wish to associate the name "SV40" with a single Bioseq for the complete genome of SV40. Let us suppose this Bioseq has the Seq-id 10. Then it is discovered that there were errors in the original Bioseq designated 10, and it is replaced by a new Bioseq from a curator with Seq-id 15. The name "SV40" can be moved to Seq-id 15 now. If a biologist wishes to see the "best" or "most typical" sequence of the SV40 genome, she would retrieve on the name "SV40". At an earlier point in time she would get Bioseq 10. At a later point she would get Bioseq 15. Note that her query is always answered in the context of best current data. On the other hand, if she had done a sequence analysis on Bioseq 10 and wanted to compare results, she would cite Seq-id 10, not the name "SV40", since her results apply to the specific Bioseq, 10, not necessarily to the "best" or "most typical" entry for the virus at the moment.

title: A Descriptive Title

A title is a brief, generally one line, description of an entry. It is extremely useful when presenting lists of Bioseqs returned from a query or search. This is the same as the familiar GenBank flatfile DEFINITION line.

Because of the utility of such terse summaries, NCBI has been experimenting with algorithmically generated titles which try to pack as much information as possible into a single line in a regular and readable format. You will see titles of this form appearing on entries produced by the NCBI journal scanning component of GenBank.

```
DEFINITION atp6=F0-ATPase subunit 6 {RNA edited} [Brassica napus=rapeseed,
 mRNA Mitochondrial, 905 nt]
DEFINITION mprA=metalloprotease, mprR=regulatory protein [Streptomyces
 coelicolor, Muller DSM3030, Genomic, 3 genes, 2040 nt]
DEFINITION pelBC gene cluster: pelB=pectate lyase isozyme B, pelC=pectate
 lyase isozyme C [Erwinia chrysanthemi, 3937, Genomic, 2481 nt]
DEFINITION glycoprotein J...glycoprotein I [simian herpes B virus SHBV,
 prototypic B virus, Genomic, 3 genes, 2652 nt]
DEFINITION glycoprotein B, gB [human herpesvirus-6 HHV6, GS, Peptide, 830
 aa]
DEFINITION {pseudogene} RESA-2=ring-infected erythrocyte surface antigen 2
 [Plasmodium falciparum, FCR3, Genomic, 3195 nt]
DEFINITION microtubule-binding protein tau {exons 4A, 6, 8 and 13/14} [human,
 Genomic, 954 nt, segment 1 of 4]
DEFINITION CAD protein carbamylphosphate synthetase domain {5' end} [Syrian
 hamsters, cell line 165-28, mRNA Partial, 553 nt]
DEFINITION HLA-DPB1 (SSK1)=MHC class II antigen [human, Genomic, 288 nt]
```

Gene and protein names come first. If both gene name and protein name are know they are linked with "=". If more than two genes are on a Bioseq then the first and last gene are given, separated by "...". A region name, if available, will precede the gene names. Extra comments will appear in {}. Organism, strain names, and molecule type and modifier appear in [] at the end. Note that the whole definition is constructed from structured information in the ASN.1 data structure by software. It is not composed by hand, but is instead a brief, machine generated summary of the entry based on data within the entry. We therefore discourage attempts to machine parse this line. It may change, but the underlying structured data will not. Software should always be designed to process the structured data.

org: What Organism Did this Come From?

If the whole Bioseq comes from a single organism (the usual case). See the Feature Table section for a detailed description of the Org-ref (organism reference) data structure.

comment: Commentary Text

A comment that applies to the whole Bioseq may go here. A comment may contain many sentences or paragraphs. A Bioseq may have many comments.

num: Applying a Numbering System to a Bioseq

One may apply a custom numbering system over the full length of the Bioseq with this Seqdescr. See the section on Numbering later in this chapter for a detailed description of the possible forms this can take. To report the numbering system used in a particular publication, the Pubdesc Seq-descr has its own Numbering slot.

maploc: Map Location

The map location given here is a Dbtag, to be able to cite a map location given by a map database to this Bioseq (e.g. "GDB", "4q21"). It is not necessarily the map location published by the author of the Bioseq. A map location published by the author would be part of a Pubdesc Seq-descr.

pir: PIR Specific Data

sp: SWISSPROT Data

embl: EMBL Data

prf: PRF Data

pdb: PDB Data

NCBI produces ASN.1 encoded entries from data provided by many different sources. Almost all of the data items from these widely differing sources are mapped into the common ASN.1 specifications described in this document. However, in all cases a small number of elements are unique to a particular data source, or cannot be unambiguously mapped into the common ASN.1 specification. Rather than lose such elements, they are carried in small data structures unique to each data source. These are specified in seqblock.asn and implemented by the C++ classes CGB_block, CEMBL_block, CSP_block, CPIR_block, CPRF_block, and CPDB_block.

genbank: GenBank Flatfile Specific Data

A number of data items unique to the GenBank flatfile format do not map readily to the common ASN.1 specification. These fields are partially populated by NCBI for Bioseqs derived from other sources than GenBank to permit the production of valid GenBank flatfile entries from those Bioseqs. Other fields are populated to preserve information coming from older GenBank entries.

pub: Description of a Publication

This Seq-descr is used both to cite a particular bibliographic source and to carry additional information about the Bioseq as it appeared in that publication, such as the numbering system to use, the figure it appeared in, a map location given by the author in that paper, and so. See the section on the Pubdesc later in this chapter for a more detailed description of this data type.

region: Name of a Genomic Region

A region of genome often has a name which is a commonly understood description for the Bioseq, such as "B-globin cluster".

user: A User-defined Structured Object

This is a place holder for software or databases to add their own structured datatypes to Bioseqs without corrupting the common specification or disabling the automatic ASN.1 syntax checking. A User-object can also be used as a feature. See the chapter on General User Objects for a detailed explanation of User-objects.

neighbors: Bioseqs Related by Sequence Similarity

NCBI computes a list of "neighbors", or closely related Bioseqs based on sequence similarity for use in the Entrez service. This descriptor is so that such context setting information could be included in a Bioseq itself, if desired.

create-date

This is the date a Bioseq was created for the first time. It is normally supplied by the source database. It may not be present when not normally distributed by the source database.

update-date

This is the date of the last update to a Bioseq by the source database. For several source databases this is the only date provided with an entry. The nature of the last update done is generally not available in computer readable (or any) form.

het: Heterogen

A "heterogen" is a non-biopolymer atom or molecule associated with Bioseqs from PDB. When a heterogen appears at the Seq-descr level, it means it was resolved in the crystal structure but is not associated with specific residues of the Bioseq. Heterogens which are associated with specific residues of the Bioseq are attached as features.

Seq-inst: Instantiating the Bioseq

Seq-inst.mol gives the physical type of the Bioseq in the living organism. If it is not certain if the Bioseq is DNA (dna) or RNA (rna), then (na) can be used to indicate just "nucleic acid". A protein is always (aa) or "amino acid". The values "not-set" or "other" are provided for internal use by editing and authoring tools, but should not be found on a finished Bioseq being sent to an analytical tool or database.

The representation class to which the Bioseq belongs is encoded in Seq-inst.repr. The values "not-set" or "other" are provided for internal use by editing and authoring tools, but should not be found on a finished Bioseq being sent to an analytical tool or database. The Data Model chapter discusses the representation class hierarchy in general. Specific details follow below.

Some of the important methods for Seq-inst are:

- IsAa - determines if the sequence type is amino acid
- IsNa - determines if the sequence type is nucleic acid

Seq-inst: Virtual Bioseq

A "virtual" Bioseq is one in which we know the type of molecule, and possibly its length, topology, and/or strandedness, but for which we do not have sequence data. It is not unusual to have some uncertainty about the length of a virtual Bioseq, so Seq-inst.fuzz may be used. The fields Seq-inst.seq-data and Seq-inst.ext are not appropriate for a virtual Bioseq.

Seq-inst: Raw Bioseq

A "raw" Bioseq does have sequence data, so Seq-inst.length must be set and there should be no Seq-inst.fuzz associated with it. Seq-inst.seq-data must be filled in with the sequence itself and a Seq-data encoding must be selected which is appropriate to Seq-inst.mol. The topology and strandedness may or may not be available. Seq-inst.ext is not appropriate.

Seq-inst: Segmented Bioseq

A segmented ("seg") Bioseq has all the properties of a virtual Bioseq, except that Seq-hist.ext of type Seq-ext.seg must be used to indicate the pieces of other Bioseqs to assemble to make the segmented Bioseq. A Seq-ext.seg is defined as a SEQUENCE OF Seq-loc, or a series of locations on other Bioseqs, taken in order.

For example, a segmented Bioseq (called "X") has a SEQUENCE OF Seq-loc which are an interval from position 11 to 20 on Bioseq "A" followed by an interval from position 6 to 15 on Bioseq "B". So "X" is a Bioseq with no internal gaps which is 20 residues long (no Seq-inst.fuzz). The first residue of "X" is the residue found at position 11 in "A". To obtain this residue, software must retrieve Bioseq "A" and examine the residue at "A" position 11. The segmented Bioseq contains no sequence data itself, only pointers to where to get the sequence data and what pieces to assemble in what order.

The type of segmented Bioseq described above might be used to represent the putative mRNA by simply pointing to the exons on two pieces of genomic sequence. Suppose however, that we had only sequenced around the exons on the genomic sequence, but wanted to represent the putative complete genomic sequence. Let us assume that Bioseq "A" is the genomic sequence of the first exon and some small amount of flanking DNA and that Bioseq "B" is the genomic sequence around the second exon. Further, we may know from mapping that the exons are separated by about two kilobases of DNA. We can represent the genomic region by creating a segmented sequence in which the first location is all of Bioseq "A". The second location will be all of a virtual Bioseq (call it "C") whose length is two thousand and which has a Seq-inst.fuzz representing whatever uncertainty we may have about the exact length of the intervening genomic sequence. The third location will be all of Bioseq "B". If "A" is 100 base pairs long and "B" is 200 base pairs, then the segmented entry is 2300 base pairs long ("A"+"C"+"B") and has the same Seq-inst.fuzz as "C" to express the uncertainty of the overall length.

A variation of the case above is when one has no idea at all what the length of the intervening genomic region is. A segmented Bioseq can also represent this case. The Seq-inst.ext location chain would be first all of "A", then a Seq-loc of type "null", then all of "B". The "null" indicates that there is no available information here. The length of the segmented Bioseq is just the sum of the length of "A" and the length of "B", and Seq-inst.fuzz is set to indicate the real length is greater-than the length given. The "null" location does not add to the overall length of the segmented Bioseq and is ignored in determining the integer value of a location on the segmented Bioseq itself. If "A" is 100 base pairs long and "B" is 50 base pairs long, then position 0 on the segmented Bioseq is equivalent to the first residue of "A" and position 100 on the segmented Bioseq is equivalent to the first residue of "B", despite the intervening "null" location indicating the gap of unknown length. Utility functions in the CSeqportUtil class can be configured to signal when crossing such boundaries, or to ignore them.

The Bioseqs referenced by a segmented Bioseq should always be from the same Seq-inst.mol class as the segmented Bioseq, but may well come from a mixture of Seq-inst.repr classes (as for example the mixture of virtual and raw Bioseq references used to describe sequenced and unsequenced genomic regions above). Other reasonable mixtures might be raw and map (see below) Bioseqs to describe a region which is fully mapped and partially sequenced, or even a mixture of virtual, raw, and map Bioseqs for a partially mapped and partially sequenced region. The "character" of any region of a segmented Bioseq is always taken from the underlying Bioseq to which it points in that region. However, a segmented Bioseq can have its own annotations. Things like feature tables are not automatically propagated to the segmented Bioseq.

Seq-inst: Reference Bioseq

A reference Bioseq is effectively a segmented Bioseq with only one pointer location. It behaves exactly like a segmented Bioseq in taking its data and "character" from the Bioseq to which it points. Its purpose is not to construct a new Bioseq from others like a segmented Bioseq, but to refer to an existing Bioseq. It could be used to provide a convenient handle to a frequently used region of a larger Bioseq. Or it could be used to develop a customized, personally annotated view of a Bioseq in a public database without losing the "live" link to the public sequence.

In the first example, software would want to be able to use the Seq-loc to gather up annotations and descriptors for the region and display them to user with corrections to align them appropriately to the sub region. In this form, a scientist my refer to the "lac region" by name, and analyze or annotate it as if it were a separate Bioseq, but each retrieve starts with a fresh copy of the underlying Bioseq and annotations, so corrections or additions made to the underlying Bioseq in the public database will be immediately visible to the scientist, without either having to always look at the whole Bioseq or losing any additional annotations the scientist may have made on the region themselves.

In the second example, software would not propagate annotations or descriptors from the underlying Bioseq by default (because presumably the scientist prefers his own view to the public one) but the connection to the underlying Bioseq is not lost. Thus the public annotations are available on demand and any new annotations added by the scientist share the public coordinate system and can be compared with those done by others.

Seq-inst: Constructed Bioseq

A constructed (const) Bioseq inherits all the attributes of a raw Bioseq. It is used to represent a Bioseq which has been constructed by assembling other Bioseqs. In this case the component Bioseqs normally overlap each other and there may be considerable redundancy of component Bioseqs. A constructed Bioseq is often also called a "contig" or a "merge".

Most raw Bioseqs in the public databases were constructed by merging overlapping gel or sequencer readings of a few hundred base pairs each. While the const Bioseq data structure can easily accommodate this information, the const Bioseq data type was not really intended for this purpose. It was intended to represent higher level merges of public sequence data and private data, such as when a number of sequence entries from different authors are found to overlap or be contained in each other. In this case a view of the larger sequence region can be constructed by merging the components. The relationship of the merge to the component Bioseqs is preserved in the constructed Bioseq, but it is clear that the constructed Bioseq is a "better" or "more complete" view of the overall region, and could replace the component Bioseqs in some views of the sequence database. In this way an author can submit a data structure to the database which in this author's opinion supersedes his own or other scientist's database entries, without the database actually dropping the other author's entries (who may not necessarily agree with the author submitting the constructed Bioseq).

The constructed Bioseq is like a raw, rather than a segmented, Bioseq because Seq-inst.seq-data must be present. The sequence itself is part of the constructed Bioseq. This is because the component Bioseqs may overlap in a number of ways, and expert knowledge or voting rules may have been applied to determine the "correct" or "best" residue from the overlapping regions. The Seq-inst.seq-data contains the sequence which is the final result of such a process.

Seq-inst.ext is not used for the constructed Bioseq. The relationship of the merged sequence to its component Bioseqs is stored in Seq-inst.hist, the history of the Bioseq (described in more detail below). Seq-hist.assembly contains alignments of the constructed Bioseq with its component Bioseqs. Any Bioseq can have a Seq-hist.assembly. A raw Bioseq may use this to show its relationship to its gel readings. The constructed Bioseq is special in that its Seq-hist.assembly shows how a high level view was constructed from other pieces. The sequence in a constructed Bioseq is only posited to exist. However, since it is constructed from data by possibly many different laboratories, it may never have been sequenced in its entirety from a single biological source.

Seq-inst: Typical or Consensus Bioseq

A consensus (consen) Bioseq is used to represent a pattern typical of a sequence region or family of sequences. There is no assertion that even one sequence exists that is exactly like this one, or even that the Bioseq is a best guess at what a real sequence region looks like. Instead it summarizes attributes of an aligned collection of real sequences. It could be a "typical" ferredoxin made by aligning ferredoxin sequences from many organisms and producing a protein sequence which is by some measure "central" to the group. By using the NCBIpaa encoding for the protein, which permits a probability to be assigned to each position that any of the standard amino acids occurs there, one can create a "weight matrix" or "profile" to define the sequence.

While a consensus Bioseq can represent a frequency profile (including the probability that any amino acid can occur at a position, a type of gap penalty), it cannot represent a regular expression per se. That is because all Bioseqs represent fixed integer coordinate systems. This property is essential for attaching feature tables or expressing alignments. There is no clear way to attach a fixed coordinate system to a regular expression, while one can approximate allowing weighted gaps in specific regions with a frequency profile. Since the consensus Bioseq is like any other, information can be attached to it through a feature table and alignments of the consensus pattern to other Bioseqs can be represented like any other alignment (although it may be computed a special way). Through the alignment, annotated features on the pattern can be related to matching regions of the aligned sequence in a straightforward way.

Seq-hist.assembly can be used in a consensus Bioseq to record the sequence regions used to construct the pattern and their relationships with it. While Seq-hist.assembly for a constructed Bioseq indicates the relationship with Bioseqs which are meant to be superseded by the constructed Bioseq, the consensus Bioseq does not in any way replace the Bioseqs in its Seq-hist.assembly. Rather it is a summary of common features among them, not a "better" or "more complete" version of them.

Seq-inst: Map Bioseqs

A map Bioseq inherits all the properties of a virtual Bioseq. For a consensus genetic map of E.coli, we can posit that the chromosome is DNA, circular, double-stranded, and about 5 million base pairs long. Given this coordinate system, we estimate the positions of genes on it based on genetic evidence. That is, we build a feature table with Gene-ref features on it (explained in more detail in the Feature Table chapter). Thus, a map Bioseq is a virtual Bioseq with a Seq-inst.ext which is a feature table. In this case the feature table is an essential part of instantiating the Bioseq, not simply an annotation on the Bioseq. This is not to say a map Bioseq cannot have a feature table in the usual sense as well. It can. It can also be used in alignments, displays, or by any software that can process or store Bioseqs. This is the great strength of this approach. A genetic or physical map is just another Bioseq and can be stored or analyzed right along with other more typical Bioseqs.

It is understood that within a particular physical or genetic mapping research project more data will have to be present than the map Bioseq can represent. But the same is true for a big sequencing project. The Bioseq is an object for reporting the result of such projects to others in a way that preserves most or all the information of use to workers outside the particular research group. It also preserves enough information to be useful to software tools within the project, such as display tools or analysis tools which were written by others.

A number of attributes of Bioseqs can make such a generic representation more "natural" to a particular research community. For the E.coli map example, above, no E.coli geneticist thinks of the positions of genes in base pairs (yet). So a Num-ref annotation (see Seq-descr, below) can be attached to the Bioseq, which provides information to convert the internal integer coordinate system of the map Bioseq to "minutes", the floating point numbers from 0.0 to 100.0 that E.coli gene positions are traditionally given in. Seq-loc objects which the Gene-ref features use to indicate their position can represent uncertainty, and thus give some idea of the accuracy of the mapping in a simple way. This representation cannot store order information directly (e.g. B and C are after A and before D, but we don't know the absolute distance and we don't know the relative order of B and C), which would need to be stored in a genetic mapping research database. However, a reasonable enough presentation can be made of this situation using locations and uncertainties to be very useful for a wide variety of purposes. As more sequence and physical map information become available, such uncertainties in gene position, at least for the "typical" chromosome, will gradually be resolved and will then map very will to such a generic model.

A physical map Bioseq has similar strengths and weaknesses as the genetic map Bioseq. It can represent an ordered map (such as an ordered restriction map) very well and easily. For some contig building approaches, ordering information is essential to the process of building the physical map and would have to be stored and processed separately by the map building research group. However, the map Bioseq serves very well as a vehicle for periodic reports of the group's best view of the physical map for consumption by the scientific public. The map Bioseq data structure maps quite well to the figures such groups publish to summarize their work. The map Bioseq is an electronic summary that can be integrated with other data and software tools.

Seq-hist: History of a Seq-inst

Seq-hist is literally the history of the Seq-inst part of a Bioseq. It does not track changes in annotation at all. However, since the coordinate system provided by the Seq-inst is the critical element for tying annotations and alignments done at various times by various people into a single consistent database, this is the most important element to track.

While Seq-hist can use any valid Seq-id, in practice NCBI will use the best available Seq-id in the Seq-hist. For this purpose, the Seq-id most tightly linked to the exact sequence itself is best. See the Seq-id discussion.

Seq-hist.assembly has been mentioned above. It is a SET OF Seq-align which show the relationship of this Bioseq to any older components that might be merged into it. The Bioseqs included in the assembly are those from which this Bioseq was made or is meant to supersede. The Bioseqs in the assembly need not all be from the author, but could come from anywhere. Assembly just sets the Bioseq in context.

Seq-hist.replaces makes an editorial statement using a Seq-hist-rec. As of a certain date, this Bioseq should replace the following Bioseqs. Databases at NCBI interpret this in a very specific way. Seq-ids in Seq-hist.replaces, which are owned by the owner of the Bioseq, are taken from the public view of the database. The author has told us to replace them with this one. If the author does not own some of them, it is taken as advice that the older entries may be obsolete, but they are not removed from the public view.

Seq-hist.replaced-by is a forward pointer. It means this Bioseq was replaced by the following Seq-id(s) on a certain date. In the case described above, that an author tells NCBI that a new Bioseq replaces some of his old ones, not only is the backward pointer (Seq-hist.replaces) provided by the author in the database, but NCBI will update the Seq-hist.replaced-by forward pointer when the old Bioseq is removed from public view. Since such old entries are still available for specific retrieval by the public, if a scientist does have annotation pointing to the old entry, the new entry can be explicitly located. Conversely, the older versions of a Bioseq can easily be located as well. Note that Seq-hist.replaced-by points only one generation forward and Seq-hist.replaces points only one generation back. This makes Bioseqs with a Seq-hist a doubly linked list over its revision history. This is very different from GenBank/EMBL/DDBJ secondary accession numbers, which only indicate "some relationship" between entries. When that relationship happens to be the replacement relationship, they still carry all accession numbers in the secondary accessions, not just the last ones, so reconstructing the entry history is impossible, even in a very general way.

Another fate which may await a Bioseq is that it is completely withdrawn. This is relatively rare but does happen. Seq-hist.deleted can either be set to just TRUE, or the date of the deletion event can be entered (preferred). If the deleted date is present, the ASN.1 will have the Date CHOICE for Seq-hist.deleted, else if the deleted boolean is TRUE the ASN.1 will have the BOOLEAN form.

Seq-data: Encoding the Sequence Data Itself

In the case of a raw or constructed Bioseq, the sequence data itself is stored in Seq-inst.seq-data, which is the data type Seq-data. Seq-data is a CHOICE of different ways of encoding the data, allowing selection of the optimal type for the case in hand. Both nucleic acid and amino acid encoding are given as CHOICEs of Seq-data rather than further subclassing first. But it is still not reasonable to encode a Bioseq of Seq-inst.mol of "aa" using a nucleic acid Seq-data type.

The Seq-data type is implemented in C++ with the CSeq_data class. This class has an E_Choice enumeration to identify the data representation scheme and a union to hold the sequence data.

The ASN.1 module seqcode.asn and C++ class CSeq_code_table define the allowed values for the various sequence encoding and the ways to display or map between codes. This permits useful information about the allowed encoding to be stored as ASN.1 data and read into a program at runtime. Some of the encodings are presented in tables in the following discussion of the different sequence encodings. The "value" is the internal numerical value of a residue in the C++ code. The "symbol" is a one letter or multi-letter symbol to be used in display to a human. The "name" is a descriptive name for the residue.

Some of the important methods for CSeq_data are:

- CSeq_data() - constructors to create objects from string or vector of char

IUPACaa: The IUPAC-IUB Encoding of Amino Acids

A set of one letter abbreviations for amino acids were suggested by the IUPAC-IUB Commission on Biochemical Nomenclature, published in J. Biol. Chem. (1968) 243: 3557-3559. It is very widely used in both printed and electronic forms of protein sequence, and many computer programs have been written to analyze data in this form internally (that is the actual ASCII value of the one letter code is used internally). To support such approaches, the IUPACaa encoding represents each amino acid internally as the ASCII value of its external one letter symbol. Note that this symbol is UPPER CASE. One may choose to display the value as lower case to a user for readability, but the data itself must be the UPPER CASE value.

In the NCBI C++ implementation, the values are stored one value per byte.

IUPACaa

Value	Symbol	Name
65	A	Alanine
66	B	Asp or Asn
67	C	Cysteine
68	D	Aspartic Acid
69	E	Glutamic Acid
70	F	Phenylalanine
71	G	Glycine
72	H	Histidine
73	I	Isoleucine
74	J	Leu or Ile
75	K	Lysine
76	L	Leucine
77	M	Methionine
78	N	Asparagine
79	O	Pyrrolysine
80	P	Proline
81	Q	Glutamine

82	R	Arginine
83	S	Serine
84	T	Threoine
86	V	Valine
87	W	Tryptophan
88	X	Undetermined or atypical
89	Y	Tyrosine
90	Z	Glu or Gln

NCBIeaa: Extended IUPAC Encoding of Amino Acids

The official IUPAC amino acid code has some limitations. One is the lack of symbols for termination, gap, or selenocysteine. Such extensions to the IUPAC codes are also commonly used by sequence analysis software. NCBI has created such a code which is simply the IUPACaa code above extended with the additional symbols.

In the NCBI C++ implementation, the values are stored one value per byte.

NCBIeaa

Value	Symbol	Name
42	*	Termination
45	-	Gap
65	A	Alanine
66	B	Asp or Asn
67	C	Cysteine
68	D	Aspartic Acid
69	E	Glutamic Acid
70	F	Phenylalanine
71	G	Glycine
72	H	Histidine
73	I	Isoleucine
74	J	Leu or Ile
75	K	Lysine
76	L	Leucine
77	M	Methionine
78	N	Asparagine
79	O	Pyrrolysine
80	P	Proline

81	Q	Glutamine
82	R	Arginine
83	S	Serine
84	T	Threoine
85	U	Selenocysteine
86	V	Valine
87	W	Tryptophan
88	X	Undetermined or atypical
89	Y	Tyrosine
90	Z	Glu or Gln

NCBIstdaa: A Simple Sequential Code for Amino Acids

It is often very useful to separate the external symbol for a residue from its internal representation as a data value. For amino acids NCBI has devised a simple continuous set of values that encompasses the set of "standard" amino acids also represented by the NCBIeaa code above. A continuous set of values means that compact arrays can be used in computer software to look up attributes for residues simply and easily by using the value as an index into the array. The only significance of any particular mapping of a value to an amino acid is that zero is used for gap and the official IUPAC amino acids come first in the list. In general, we recommend the use of this encoding for standard amino acid sequences.

In the NCBI C++ implementation, the values are stored one value per byte.

NCBIstdaa

Value	**Symbol**	**Name**
0	-	Gap
1	A	Alanine
2	B	Asp or Asn
3	C	Cysteine
4	D	Aspartic Acid
5	E	Glutamic Acid
6	F	Phenylalanine
7	G	Glycine
8	H	Histidine
9	I	Isoleucine
10	K	Lysine
11	L	Leucine
12	M	Methionine

13	N	Asparagine
14	P	Proline
15	Q	Glutamine
16	R	Arginine
17	S	Serine
18	T	Threoine
19	V	Valine
20	W	Tryptophan
21	X	Undetermined or atypical
22	Y	Tyrosine
23	Z	Glu or Gln
24	U	Selenocysteine
25	*	Termination
26	O	Pyrrolysine
27	J	Leu or Ile

NCBI8aa: An Encoding for Modified Amino Acids

Post-translational modifications can introduce a number of non-standard or modified amino acids into biological molecules. The NCBI8aa code will be used to represent up to 250 possible amino acids by using the remaining coding space in the NCBIstdaa code. That is, for the first 26 values, NCBI8aa will be identical to NCBIstdaa. The remaining 224 values will be used for the most commonly encountered modified amino acids. Only the first 250 values will be used to signify amino acids, leaving values in the range of 250-255 to be used for software control codes. Obviously there are a very large number of possible modified amino acids, especially if one takes protein engineering into account. However, the intent here is to only represent commonly found biological forms. This encoding is not yet available since decisions about what amino acids to include not all have been made yet.

NCBIpaa: A Profile Style Encoding for Amino Acids

The NCBIpaa encoding is designed to accommodate a frequency profile describing a protein motif or family in a form which is consistent with the sequences in a Bioseq. Each position in the sequence is defined by 30 values. Each of the 30 values represents the probability that a particular amino acid (or gap, termination, etc.) will occur at that position. One can consider each set of 30 values an array. The amino acid for each cell of the 30 value array corresponds to the NCBIstdaa index scheme. This means that currently only the first 26 array elements will ever have a meaningful value. The remaining 4 cells are available for possible future additions to NCBIstdaa. Each cell represents the probability that the amino acid defined by the NCBIstdaa index to that cell will appear at that position in the motif or protein. The probability is encoded as an 8-bit value from 0-255 corresponding to a probability from 0.0 to 1.0 by interpolation.

This type of encoding would presumably never appear except in a Bioseq of type "consensus". In the C++ implementation these amino acids are encoded at 30 bytes per amino acid in a simple linear order. That is, the first 30 bytes are the first amino acid, the second 30 the next amino acid, and so on.

IUPACna: The IUPAC-IUB Encoding for Nucleic Acids

Like the IUPACaa codes the IUPACna codes are single letters for nucleic acids and the value is the same as the ASCII value of the recommended IUPAC letter. The IUPAC recommendations for nucleic acid codes also include letters to represent all possible ambiguities at a single position in the sequence except a gap. To make the values non-redundant, U is considered the same as T. Whether a sequence actually contains U or T is easily determined from Seq-inst.mol. Since some software tools are designed to work directly on the ASCII representation of the IUPAC letters, this representation is provided. Note that the ASCII values correspond to the UPPER CASE letters. Using values corresponding to lower case letters in Seq-data is an error. For display to a user, any readable case or font is appropriate.

The C++ implementation encodes one value for a nucleic acid residue per byte.

IUPACna

Value	Symbol	Name
65	A	Adenine
66	B	G or T or C
67	C	Cytosine
68	D	G or A or T
71	G	Guanine
72	H	A or C or T
75	K	G or T
77	M	A or C
78	N	A or G or C or T
82	R	G or A
83	S	G or C
84	T	Thymine
86	V	G or C or A
87	W	A or T
89	Y	T or C

NCBI4na: A Four Bit Encoding of Nucleic Acids

It is possible to represent the same set of nucleic acid and ambiguities with a four bit code, where one bit corresponds to each possible base and where more than one bit is set to represent ambiguity. The particular encoding used for NCBI4na is the same as that used on the GenBank Floppy Disk Format. A four bit encoding has several advantages over the direct mapping of the ASCII IUPAC codes. One can represent "no base" as 0000. One can match various ambiguous or unambiguous bases by a simple AND. For example, in NCBI4na 0001=A, 0010=C, 0100=G, 1000=T/U. Adenine (0001) then matches Purine (0101) by the AND method. Finally, it is possible to store the sequence in half the space by storing two bases per byte. This is done both in the ASN.1 encoding and in the NCBI C++ software implementation.

Utility functions (see the CSeqportUtil class) allow the developer to ignore the complexities of storage while taking advantage of the greater packing. Since nucleic acid sequences can be very long, this is a real savings.

NCBI4na

Value	**Symbol**	**Name**
0	-	Gap
1	A	Adenine
2	C	Cytosine
3	M	A or C
4	G	Guanine
5	R	G or A
6	S	G or C
7	V	G or C or A
8	T	Thymine/Uracil
9	W	A or T
10	Y	T or C
11	H	A or C or T
12	K	G or T
13	D	G or A or T
14	B	G or T or C
15	N	A or G or C or T

NCBI2na: A Two Bit Encoding for Nucleic Acids

If no ambiguous bases are present in a nucleic acid sequence it can be completely encoded using only two bits per base. This allows encoding into ASN.1 or storage in the NCBI C++ implementation with a four fold savings in space. As with the four bit packing, the CSeqportUtil class allows the programmer to ignore the complexities introduced by the packing. The two bit encoding selected is the same as that proposed for the GenBank CDROM.

NCBI2na

Value	**Symbol**	**Name**
0	A	Adenine
1	C	Cytosine
2	G	Guanine
3	T	Thymine/Uracil

NCBI8na: An Eight Bit Sequential Encoding for Modified Nucleic Acids

The first 16 values of NCBI8na are identical with those of NCBI4na. The remaining possible 234 values will be used for common, biologically occurring modified bases such as those found in tRNAs. This full encoding is still being determined at the time of this writing. Only the first 250 values will be used, leaving values in the range of 250-255 to be used as control codes in software.

NCBIpna: A Frequency Profile Encoding for Nucleic Acids

Frequency profiles have been used to describe motifs and signals in nucleic acids. This can be encoded by using five bytes per sequence position. The first four bytes are used to express the probability that particular bases occur at that position, in the order A, C, G, T as in the NCBI2na encoding. The fifth position encodes the probability that a base occurs there at all. Each byte has a value from 0-255 corresponding to a probability from 0.0-1.0.

The sequence is encoded as a simple linear sequence of bytes where the first five bytes code for the first position, the next five for the second position, and so on. Typically the NCBIpna notation would only be found on a Bioseq of type consensus. However, one can imagine other uses for such an encoding, for example to represent knowledge about low resolution sequence data in an easily computable form.

Tables of Sequence Codes

Various sequence alphabets can be stored in tables of type Seq-code-table, defined in seqcode.asn. An enumerated type, Seq-code-type is used as a key to each table. Each code can be thought of as a square table essentially like those presented above in describing each alphabet. Each "residue" of the code has a numerical one-byte value used to represent that residue both in ASN.1 data and in internal C++ structures. The information necessary to display the value is given by the "symbol". A symbol can be in a one-letter series (e.g. A,G,C,T) or more than one letter (e.g. Met, Leu, etc.). The symbol gives a human readable representation that corresponds to each numerical residue value. A name, or explanatory string, is also associated with each.

So, the NCBI2na code above would be coded into a Seq-code-table very simply as:

```
{ -- NCBI2na
 code ncbi2na ,
 num 4 , -- continuous 0-3
 one-letter TRUE , -- all one letter codes
 table {
 { symbol "A", name "Adenine" },
 { symbol "C", name "Cytosine" },
 { symbol "G", name "Guanine" },
 { symbol "T", name "Thymine/Uracil"}
 } , -- end of table
 comps { -- complements
 3,
 2,
 1,
 0
 }
} ,
```

The table has 4 rows (with values 0-3) with one letter symbols. If we wished to represent a code with values which do not start at 0 (such as the IUPAC codes) then we would set the OPTIONAL "start-at" element to the value for the first row in the table.

In the case of nucleic acid codes, the Seq-code-table also has rows for indexes to complement the values represented in the table. In the example above, the complement of 0 ("A") is 3 ("T").

Mapping Between Different Sequence Alphabets

A Seq-map-table provides a mapping from the values of one alphabet to the values of another, very like the way complements are mapped above. A Seq-map-table has two Seq-code-types, one giving the alphabet to map from and the other the alphabet to map to. The Seq-map-table has the same number of rows and the same "start-at" value as the Seq-code-table for the alphabet it maps FROM. This makes the mapping a simple array lookup using the value of a residue of the FROM alphabet and subtracting "start-at". Remember that alphabets are not created equal and mapping from a bigger alphabet to a smaller may result in loss of information.

Pubdesc: Publication Describing a Bioseq

A Pubdesc is a data structure used to record how a particular publication described a Bioseq. It contains the citation itself as a Pub-equiv (see the Bibliographic References chapter) so that equivalent forms of the citation (e.g. a MEDLINE uid and a Cit-Art) can all be accommodated in a single data structure. Then a number of additional fields allow a more complete description of what was presented in the publication. These extra fields are generally only filled in for entries produced by the NCBI journal scanning component of GenBank, also known as the Backbone database. This information is not generally available in data from any other database yet.

Pubdesc.name is the name given the sequence in the publication, usually in the figure. Pubdesc.fig gives the figure the Bioseq appeared in so a scientist can locate it in the paper. Pubdesc.num preserves the numbering system used by the author (see Numbering below). Pubdesc.numexc, if TRUE, indicates that a "numbering exception" was found (i.e. the author's numbering did not agree with the number of residues in the sequence). This usually indicates an error in the preparation of the figure. If Pubdesc.poly-a is TRUE, then a poly-A tract was indicated for the Bioseq in the figure, but was not explicitly preserved in the sequence itself (e.g. ...AGAATTTCT (Poly-A)). Pubdesc.maploc is the map location for this sequence as given by the author in this paper. Pubdesc.seq-raw allows the presentation of the sequence exactly as typed from the figure. This is never used now. Pubdesc.align-group, if present, indicates the Bioseq was presented in a group aligned with other Bioseqs. The align-group value is an arbitrary integer. Other Bioseqs from the same publication which are part of the same alignment will have the same align-group number.

Pubdesc.comment is simply a free text comment associated with this publication. SWISSPROT entries may also have this field filled.

Numbering: Applying a Numbering System to a Bioseq

Internally, locations on Bioseqs are ALWAYS integer offsets in the range 0 to (length - 1). However, it is often helpful to display some other numbering system. The Numbering data structure supports a variety of numbering styles and conventions. In the ASN.1 specification, it is simply a CHOICE of the four possible types. When a Numbering object is supplied as a Seq-descr, then it applies to the complete length of the Bioseq. A Numbering object can also be a feature, in which case it only applies to the interval defined by the feature's location.

Num-cont: A Continuous Integer Numbering System

The most widely used numbering system for sequences is some form of a continuous integer numbering. Num-cont.refnum is the number to assign to the first residue in the Bioseq. If Num-cont.has-zero is TRUE, the numbering system uses zero. When biologists start numbering with a negative number, it is quite common for them to skip zero, going directly from -1 to +1, so the DEFAULT for has-zero is FALSE. This only reflects common usage, not any recommendation in terms of convention. Any useful software tool should support both conventions, since they are both used in the literature. Finally, the most common numbering systems are ascending; however descending numbering systems are encountered from time to time, so Num-cont.ascending would then be set to FALSE.

Num-real: A Real Number Numbering Scheme

Genetic maps may use real numbers as "map units" since they treat the chromosome as a continuous coordinate system, instead of a discrete, integer coordinate system of base pairs. Thus a Bioseq of type "map" which may use an underlying integer coordinate system from 0 to 5 million may be best presented to the user in the familiar 0.0 to 100.0 map units. Num-real supports a simple linear equation specifying the relationship:

map units = (Num-real.a * base_pair_position) + Num-real.b

in this example. Since such numbering systems generally have their own units (e.g. "map units", "centisomes", "centimorgans", etc), Num-real.units provides a string for labeling the display.

Num-enum: An Enumerated Numbering Scheme

Occasionally biologists do not use a continuous numbering system at all. Crystallographers and immunologists, for example, who do extensive studies on one or a few sequences, may name the individual residues in the sequence as they fit them into a theoretical framework. So one might see residues numbered ... "10" "11" "12" "12A" "12B" "12C" "13" "14" ... To accommodate this sort of scheme the "name" of each residue must be explicitly given by a string, since there is no anticipating any convention that may be used. The Num-enum.num gives the number of residue names (which should agree with the number of residues in the Bioseq, in the case of use as a Seq-descr), followed by the names as strings.

Num-ref: Numbering by Reference to Another Bioseq

Two types of references are allowed. The "sources" references are meant to apply the numbering system of constituent Bioseqs to a segmented Bioseq. This is useful for seeing the mapping from the parts to the whole.

The "aligns" reference requires that the Num-ref-aligns alignment be filled in with an alignment of the target Bioseq with one or more pieces of other Bioseqs. The numbering will come from the aligned pieces.

Numbering: C++ Class

A Numbering object is implemented by the C++ class CNumbering. The choice of numbering type is represented by the E_Choice enumeration. The class contains methods for getting and setting the various types of numbering.

Collections of Sequences

This section describes the types used to organize multiple Bioseqs into tree structures. The types are located in the seqset.asn module.

C++ Implementation Notes

Introduction

A biological sequence is often most appropriately stored in the context of other, related sequences. Such a collection might have a biological basis (e.g. a nucleic acid and its translated proteins, or the chains of an enzyme complex) or some other basis (e.g. a release of GenBank, or the sequences published in an article). The Bioseq-set provides a framework for collections of sequences.

Seq-entry: The Sequence Entry

Sometimes a sequence is not part of a collection (e.g. a single annotated protein). Thus a sequence entry could be either a single Bioseq or a collection of them. A Seq-entry is an entity which represents this choice. A great deal of NCBI software is designed to accept a Seq-entry as the primary unit of data. This is the most powerful and flexible object to use as a target software development in general.

Some of the important methods for CSeq_entry are:

- GetLabel() - append a label based on type or content of the current Seq-entry
- GetParentEntry() - gets the parent of the current Seq-entry
- Parentize() - recursive update of parent Seq-entries

Bioseq-set: A Set Of Seq-entry's

A Bioseq-set contains a convenient collection of Seq-entry's. It can have descriptors and annotations just like a single Bioseq (see Biological Sequences). It can have identifiers for the set, although these are less thoroughly controlled than Seq-ids at this time. Since the "heart" of a Bioseq-set is a collection of Seq-entry's, which themselves are either a Bioseq or a Bioseq-set, a Bioseq-set can recursively contain other sets. This recursive property makes for a very rich data structure, and a necessary one for biological sequence data, but presents new challenges for software to manipulate and display it. We will discuss some guidelines for building and using Bioseq-sets below, based on the NCBI experience to date.

Some of the important methods for CBioseq_set are:

- GetLabel() - append a label based on type or content of CBioseq_set
- GetParentSet() - gets the parent of the current CBioseq_set

id: local identifier for this set

The id field just contains an integer or string to identify this set for internal use by a software system or database. This is useful for building collections of sequences for temporary use, but still be able to cite them.

coll: global identifier for this set

The coll field is a Dbtag, which will accept a string to identify a source database and a string or integer as an identifier within that database. This semi-controlled form provides a global identifier for the set of sequences in a simple way.

level: nesting level of set

Since Bioseq-sets are recursive, the level integer was conceived as a way of explicitly indicating the nesting level. In practice we have found this to be of little or no use and recommend it be ignored and eventually removed.

class: classification of sets

The class field is an attempt to classify sets of sequences that may be widely used. There are a number which are just releases of well known databases and others which represent biological groupings.

Bioseq-set classes

The following table summarizes the types of Bioseq-sets:

Value	ASN.1 name	Explanation
0	not-set	not determined
1	nuc-prot	a nucleic acid and the proteins from its coding regions
2	segset	a segmented Bioseq and the Bioseqs it is made from
3	conset	a constructed Bioseq and the Bioseqs it was assembled from
4	parts	a set cotained within a segset or conset holding the Bioseqs which are the components of the segmented or constructed Bioseq
5	gibb	GenInfo Backbone entries (NCBI Journal Scanning Database)
6	gi	GenInfo entries (NCBI ID Database)
7	genbank	GenBank entries
8	pir	PIR entries
9	pub-set	all the Seq-entry's from a single publication
10	equiv	a set of equivalent representations of the same sequence (e.g. a genetic map Bioseq and a physical map Bioseq for the same chromosome)
11	swissprot	SWISSPROT entries
12	pdb-entry	all the Bioseqs associated with a single PDB structure
255	other	new type. Usually Bioseq-set.release will have an explanatory string

release: an explanatory string

This is just a free text field which can contain a human readable description of the set. Often used to show which release of GenBank, for example.

date

This is a date associated with the creation of this set.

descr: Seq-descr for this set

Just like a Bioseq, a Bioseq-set can have Seq-descr (see Biological Sequences) which set it in a biological or bibliographic context, or confer a title or a name. The rule for descriptors at the set level is that they apply to "all of everything below". So if an Org-ref is given at the set level, it means that every Bioseq in the set comes from that organism. If this is not true, then Org-ref would not appear on the set, but different Org-refs would occur on lower level members.

For any Bioseq in arbitrarily deeply nested Bioseq-sets, one should be able to collect all Bioseq-set.descr from all higher level Bioseq-sets that contain the Bioseq, and move them to the Bioseq. If this process introduces any confusion or contradiction, then the set level descriptor has been incorrectly used.

The only exception to this is the title and name types, which often refer to the set level on which they are placed (a nuc-prot may have the title "Adh gene and ADH protein", while the Bioseqs have the titles "Adh gene" and "ADH protein". The gain in code sharing by using exactly the same Seq-descr for Bioseq or Bioseq-set seemed to outweigh the price of this one exception to the rule.

To simplify access to elements like this that depend on a set context, a series of BioseqContext () functions are provided in utilities which allow easy access to all relevant descriptors starting with a specific Bioseq and moving up the levels in the set.

seq-set: the sequences and sets within the Bioseq-set

The seq-set field contains a SEQUENCE OF Seq-entry which represent the contents of the Bioseq-set. As mentioned above, these may be nested internally to any level. Although there is no guarantee that members of a set will come in any particular order, NCBI finds the following conventions useful and natural.

For sets of entries from specific databases, each Seq-entry is the "natural" size of an entry from those databases. Thus GenBank will contain a set of Seq-entry which will be a mixture of Bioseq (just a nucleic acid, no coding regions), seg-set (segmented nucleic acid, no coding regions), or nuc-prot (nucleic acid (as Bioseq or seg-set) and proteins from the translated coding regions). PDB will contain a mixture of Bioseq (single chain structures) or pdb-entry (multi-chain structures).

A segset, representing a segmented sequence combines the segmented Bioseq with the set of the Bioseqs that make it up.

```
segset (Bioseq-set) contains
 segmented sequence (Bioseq)
 parts (Bioseq-set) contains
 first piece (Bioseq)
 second piece (Bioseq
 etc
```

A consset has the same layout as a segset, except the top level Bioseq is constructured rather than segmented.

A nuc-prot set gives the nucleic acid and its protein products at the same levels.

```
nuc-prot (Bioseq-set) contains
 nucleic acid (Bioseq)
 protein1 (Bioseq)
 protein2 (Bioseq)
 etc.
```

A nuc-prot set where the nucleic acid is segmented simply replaces the nucleic acid Bioseq with a seg-set.

```
nuc-prot (Bioseq-set) contains
 nucleic acid segset (Bioseq-set) contains
 segmented sequence (Bioseq)
 parts (Bioseq-set) contains
 first piece (Bioseq)
 second piece (Bioseq
 etc
 protein1 (Bioseq)
 protein2 (Bioseq)
 etc.
```

annot: Seq-annots for the set

A Bioseq-set can have Seq-annots just like a Bioseq can. Because all forms of Seq-annot use explicit ids for the Bioseqs they reference, there is no dependence on context. Any Seq-annot can appear at any level of nesting in the set (or even stand alone) without any loss of information.

However, as a convention, NCBI puts the Seq-annot at the nesting level of the set that contains all the Bioseqs referenced by it, if possible. So if a feature applies just to one Bioseq, it goes in the Bioseq.annot itself. If it applies to all the members of a segmented set, it goes in Bioseq-set.annot of the segset. If, like a coding region, it points to both nucleic acid and protein sequences, it goes in the Bioseq-set.annot of the nuc-prot set.

Bioseq-sets are Convenient Packages

Remember that Bioseq-sets are just convenient ways to package Bioseqs and associated annotations. But Bioseqs may appear in various contexts and software should always be prepared to deal with them that way. A segmented Bioseq may not appear as part of a segset and a Bioseq with coding regions may not appear as part of a nuc-prot set. In both cases the elements making up the segmented Bioseq and the Bioseqs involved in the coding regions all use Seq-locs, which explicit reference Seq-ids. So they are not dependent on context. NCBI packages Bioseqs in sets for convenience, so all the closely related elements can be retrieved together. But this is only a convenience, not a requirement of the specification. The same caveat applies to the ordering conventions within a set, described above.

Sequence Locations and Identifiers

This section contains documentation for types used to identify Bioseqs and describe locations on them. These types are defined in the seqloc.asn module.

C++ Implementation Notes

- Introduction
- Seq-id: Identifying Sequences
- Seq-id: Semantics of Use
- Seq-id: The C++ Implementation

- NCBI ID Database: Imposing Stable Seq-ids
- Seq-loc: Locations on a Bioseq
- Seq-loc: The C++ Implementation
- ASN.1 Specification: seqloc.asn

Introduction

As described in the Biological Sequences chapter, a Bioseq always has at least one identifier. This means that any valid biological sequence can be referenced by using this identifier. However, all identifiers are not created equal. They may differ in their basic structure (e.g. a GenBank accession number is required to have an uppercase letter followed by exactly five digits while the NCBI GenInfo Id uses a simple integer identifier). They also differ in how they are used (e.g. the sequence identified by the GenBank accession number may change from release to release while the sequence identified by the NCBI GenInfo Id will always be exactly the same sequence).

Locations of regions on Bioseqs are always given as integer offsets, also described in the Biological Sequences chapter. So the first residue is always 0 and the last residue is always (length - 1). Further, since all the classes of Bioseqs from bands on a gel to genetic or physical maps to sequenced DNA use the same integer offset convention, locations always have the same form and meaning even when moving between very different types of Bioseq representations. This allows alignment, comparison, and display functions, among others, to have the same uniform interface and semantics, no matter what the underlying Bioseq class. Specialized numbering systems are supported but only as descriptive annotation (see Numbering in Biological Sequences and Feature types "seq" and "num" in Sequence Features). The internal conventions for positions on sequences are always the same.

There are no implicit Bioseq locations. All locations include a sequence identifier. This means Features, Alignments, and Graphs are always independent of context and can always be exchanged, submitted to databases, or stored as independent objects. The main consequence of this is that information ABOUT regions of Bioseqs can be developed and contributed to the public scientific discussion without any special rights of editing the Bioseq itself needing to be granted to anyone but the original author of the Bioseq. Bioseqs in the public databases, then, no longer need an anointed curator (beyond the original author) to be included in ongoing scientific discussion and data exchange by electronic media.

In addition to the various sequence location and identifier classes, several convenience functions for comparing or manipulating Na-strands are defined in Na_strand.hpp:

- IsForward()
- IsReverse()
- Reverse()
- SameOrientation()

Seq-id: Identifying Sequences

In a pure sense, a Seq-id is meant to unambiguously identify a Bioseq. Unfortunately, different databases have different semantic rules regarding the stability and ambiguity of their best available identifiers. For this reason a Bioseq can have more than one Seq-id, so that the Seq-id with the best semantics for a particular use can be selected from all that are available for that Bioseq, or so that a new Seq-id with different semantics can be conferred on an existing Bioseq. Further, Seq-id is defined as a CHOICE of datatypes which may differ considerably in their structure and semantics from each other. Again, this is because differing sequence databases use different conventions for identifying sequences and it is important not to lose this critical information from the original data source.

One Seq-id type, "gi", has been implemented specifically to make a simple, absolutely stable Seq-id available for sequence data derived from any source. It is discussed in detail below.

A Textseq-id structure is used in many Seq-ids described below. It has four possible fields; a name, an accession number, a release, and a version. Formally, all fields are OPTIONAL, although to be useful, a Textseq-id should have at least a name or an accession or both. This style of Seq-id is used by GenBank, EMBL, DDBJ, PIR, SWISS-PROT, and PRF, but the semantics of its use differ considerably depending on the database. However none of these databases guarantees the stability of name or accession (i.e. that it points at a specific sequence), so to be unambiguous the id must also have the version. See the discussion under Seq-id: Semantics for details.

Some important methods of the CSeq_id class are:

- CSeq_id() -- constructors to simplify creation of Seq-ids from primitive types (string, int). Some of these constructors auto-detect the type of the Seq-id from its string representation.
- Compare() -- compare Seq-ids.
- GetTextseq_Id () -- checks whether the Seq-id subtype is Textseq-id compatible and returns its value.
- IdentifyAccession() -- deduces Seq-id information from a bare accession.
- Match() -- compare Seq-ids.

Some important nonmember template functions are:

- FindGi() -- returns gi from id list if exists, returns 0 otherwise.
- FindTextseq_id() -- returns text seq-id from id list if exists, returns 0 otherwise.
- GetSeq_idByType() -- search the container of CRef<CSeq_id> for the id of given type.

Seq-id: Semantics of Use

Different databases use their ids in different ways and these patterns may change over time. An attempt is made is this section to describe current usage and offer some guidelines for interpreting Seq-ids.

local: Privately Maintained Data

The local Seq-id is an Object-id (see discussion in General Use Objects), which is a CHOICE of a string or an integer. This is to reconcile the requirement that all Bioseqs have a Seq-id and the needs of local software tools to manipulate data produced or maintained privately. This might be pre-publication data, data still being developed, or proprietary data. The Object-id will accommodate either a string or a number as is appropriate for the local environment. It is the responsibility of local software to keep the local Seq-ids unique. A local Seq-id is not globally unique, so when Bioseqs with such identifiers are published or exchanged, context (i.e. the submitter or owner of the id) must be maintained or a new id class must be applied to the Bioseq (e.g. the assignment of a GenBank accession upon direct data submission to GenBank).

refseq: From the Reference Sequence project at the NCBI

The Reference Sequence project at the NCBI aims to provide a comprehensive, integrated, non-redundant, well-annotated set of sequences, including genomic DNA, transcripts, and proteins. RefSeq assigns accessions (but not a LOCUS) to all entries. RefSeq accessions begin with two letters followed by an underscore, with additional letters after the underscore for some accessions. The leading characters have a distinct meaning as described in the RefSeq accession format reference.

general: Ids from Local Databases

The Seq-id type "general" uses a Dbtag (see discussion in General Use Objects), which is an Object-id as in Seq-id.local, above, with an additional string to identify a source database. This means that an integer or string id from a smaller database can create Seq-ids which both cite the database source and make the local Seq-ids globally unique (usually). For example, the EcoSeq database is a collection of E.coli sequences derived from many sources, maintained by Kenn Rudd. Each sequence in EcoSeq has a unique descriptive name which is used as its primary identifier. A "general" Seq-id could be make for the EcoSeq entry "EcoAce" by making the following "general" Seq-id:

```
Seq-id ::= general {
 db "EcoSeq" ,
 tag str "EcoAce" }
```

gibbsq, gibbmt: GenInfo Backbone Ids

The "gibbsq" and "gibbmt" IDs were formerly used to access the "GenInfo Backbone" database. They are now obsolete.

genbank, embl, ddbj: The International Nucleic Acid Sequence Databases

NCBI (GenBank) in the U.S., the European Molecular Biology Laboratory datalibrary (EMBL) in Europe, and the DNA Database of Japan (DDBJ) in Japan are members of an international collaboration of nucleic acid sequence databases. Each collects data, often directly submitted by authors, and makes releases of its data in its own format independently of each other. However, there are agreements in place for all the parties to exchange information with each other in an attempt to avoid duplication of effort and provide a world wide comprehensive database to their users. So a release by one of these databases is actually a composite of data derived from all three sources.

All three databases assign a mnemonic name (called a LOCUS name by GenBank and DDBJ, and an entry name by EMBL) which is meant to carry meaning encoded into it. The first few letters indicate the organism and next few a gene product, and so on. There is no concerted attempt to keep an entry name the same from release to release, nor is there any attempt for the same entry to have the same entry name in the three different databases (since they construct entry names using different conventions). While many people are used to referring to entries by name (and thus name is included in a Textseq-id) it is a notoriously unreliable way of identifying a Bioseq and should normally be avoided.

All three databases also assign an Accession Number to each entry. Accession numbers do not convey meaning, other than in a bookkeeping sense. Unlike names, accession numbers are meant to be same for the same entry, no matter which database one looks in. Thus, accession number is the best id for a Bioseq from this collaboration. Unfortunately rules for the use of accession numbers have not required that an accession number uniquely identify a sequence. A database may change an accession when it merely changes the annotation on an entry. Conversely, a database may not change an accession even though it has changed the sequence itself. There is no consistency about when such events may occur. There is also no exact method of recording the history of an entry in this collaboration, so such accession number shifts make it possible to lose track of entries by outside users of the databases. With all these caveats, accession numbers are still the best identifiers available within this collaboration.

To compensate for such shifts, it is advisable to supplement accession numbers with version numbers to yield stable, unique identifiers for all three databases. (Historically, it was likewise possible to supplement them with release fields, but those are no longer in active use and retrieval services will disregard them.)

pir: PIR International

The PIR database is also produced through an international collaboration with contributors in the US at the Protein Identification Resource of the National Biomedical Research Foundation (NBRF), in Europe at the Martinsried Institute for Protein Sequences (MIPS), and in Japan at the International Protein Information Database in Japan (JIPID). They also use an entry name and accession number. The PIR accession numbers, however, are not related to the GenBank/EMBL/DDBJ accession numbers in any way and have a very different meaning. In PIR, the entry name identifies the sequence, which is meant to be the "best version" of that protein. The accession numbers are in transition from a meaning more similar to the GenBank/EMBL/DDBJ accessions, to one in which an accession is associated with protein sequences exactly as they appeared in specific publications. Thus, at present, PIR ids may have both an accession and a name, they will move to more typically having either a name or an accession, depending on what is being cited, the "best" sequence or an original published sequence.

swissprot: UniProt Knowledgebase

Originally the Seq-id type "swissprot" referred to the Swiss-Prot database, but now it refers to the UniProtKB database. This change was made after the Swiss-Prot, TrEMBL, and PIR databases were combined to form UniProtKB. The swissprot name is meant to be easily remembered and it codes biological information, but it is not a stable identifier from release to release. The accession, which only conveys bookkeeping information, serves as a relatively stable identifier from release to release, and in conjunction with a version uniquely identifies a UniProtKB entry.

With the exception of legacy PIR entry names (which the C++ Toolkit cannot recognize when untagged), UniProtKB identifiers are coordinated with those of GenBank, EMBL, and DDBJ and do not conflict.

prf: Protein Research Foundation

The Protein Research Foundation in Japan has a large database of protein sequence and peptide fragments derived from the literature. Again, there is a name and an accession number. Since this database is meant only to record the sequence as it appeared in a particular publication, the relationship between the id and the sequence is quite stable in practice.

patent: Citing a Patent

The minimal information to unambiguously identify a sequence in a patent is first to unambiguously identify the patent (by the Patent-seq-id.cit, see Bibliographic References for a discussion of Id-pat) and then providing an integer serial number to identify the sequence within the patent. The sequence data for sequence related patents are now being submitted to the international patent offices in computer readable form, and the serial number for the sequence is assigned by the processing office. However, older sequence related patents were not assigned serial numbers by the processing patent offices. For those sequences the serial number is assigned arbitrarily (but still uniquely). Note that a sequence with a Patent-seq-id just appeared as part of a patent document. It is NOT necessarily what was patented by the patent document.

pdb: Citing a Biopolymer Chain from a Structure Database

The Protein Data Bank (PDB, also known as the Brookhaven Database), is a collection of data about structures of biological entities such hemoglobin or cytochrome c. The basic entry in PDB is a structural model of a molecule, not a sequence as in most sequence databases. A molecule may have multiple chains. So a PDB-seq-id has a string for the PDB entry name (called PDB-mol-id here) and a single character for a chain identifier within the molecule. The use of the single character just maps the PDB practice. The character may be a digit, a letter, or even a space (ASCII 32). As with the databases using the Textseq-id, the sequence of the chain in PDB associated with this information is not stable, so to be unambiguous the id must also include the release date.

giim: GenInfo Import Id

A Giimport-id ("giim") was a temporary id used to identify sequences imported into the GenInfo system at NCBI before long term identifiers, such as "gi", became stable. It is now obsolete.

gi: A Stable, Uniform Id Applied to Sequences From All Sources

A Seq-id of type "gi" is a simple integer assigned to a sequence by the NCBI "ID" database. It can be applied to a Bioseq of any representation class, nucleic acid or protein. It uniquely identifies a sequence from a particular source. If the sequence changes at all, then a new "gi" is assigned. The "gi" does not change if only annotations are changed. Thus the "gi" provides a simple, uniform way of identifying a stable coordinate system on a Bioseq provided by data sources which themselves may not have stable ids. This is the identifier of choice for all references to Bioseqs through features or alignments. See discussion below.

Seq-id: The C++ Implementation

A Seq-id is implemented in C++ as a choice, summarized in the following table:

Seq-id

Value	**Enum name**	**Description**
0	e_not_set	no variant selected
1	e_Local	local use
2	e_Gibbsq	GenInfo back-bone seq id
3	e_Gibbmt	GenInfo back-bone molecule
4	e_Giim	GenInfo import id
5	e_Genbank	genbank
6	e_Embl	embl
7	e_Pir	pir
8	e_Swissprot	swissprot
9	e_Patent	patent
10	e_Other	for historical reasons, 'other' = 'refseq'

11	e_General	general - for other databases
12	e_Gi	GenInfo integrated database
13	e_Ddbj	DDBJ
14	e_Prf	PRF SEQDB
15	e_Pdb	PDB sequence
16	e_Tpg	3rd party annot/seq Genbank
17	e_Tpeq	3rd party annot/seq EMBL
18	e_Tpd	3rd party annot/seq DDBJ
19	e_Gpipe	Internal NCBI genome pipeline processing id
20	e_Named_annot_track	Internal named annotation tracking id

A large number of additional functions for manipulating SeqIds are described in the Sequence Utilities chapter.

NCBI ID Database: Imposing Stable Seq-ids

As described in the Data Model section, Bioseqs provide a simple integer coordinate system through which a host of different data and analytical results can be easily associated with each other, even with scientists working independently of each other and on heterogeneous systems. For this model to work, however, requires stable identifiers for these integer coordinate systems. If one scientist notes a coding region from positions 10-50 of sequence "A", then the database adds a single base pair at position 5 of "A" without changing the identifier of "A", then at the next release of the database the scientist's coding region is now frame-shifted one position and invalid. Unfortunately this is currently the case due to the casual use of sequence identifiers by most existing databases.

Since NCBI integrates data from many different databases which follow their own directions, we must impose stable ids on an unstable starting material. While a daunting task, it is not, in the main, impossible. We have built a database called "ID", whose sole task is to assign and track stable sequence ids. ID assigns "gi" numbers, simple arbitrary integers which stably identify a particular sequence coordinate system.

The first time ID "sees" a Bioseq, say EMBL accession A00000, it checks to see if it has a Bioseq from EMBL with this accession already. If not, it assigns a new GI, say 5, to the entry and adds it to the Bioseq.id chain (the original EMBL id is not lost). It also replaces all references in the entry (say in the feature table) to EMBL A00000 to GI 5. This makes the annotations now apply to a stable coordinate system.

Now EMBL sends an update of the entry which is just a correction to the feature table. The same process occurs, except this time there is a previous entry with the same EMBL accession number. ID retrieves the old entry and compares the sequence of the old entry with the new entry. Since they are identical it reassigns GI 5 to the same entry, converts the new annotations, and stores it as the most current view of that EMBL entry.

Now ID gets another update to A00000, but this time the sequence is different. ID assigns a new GI, say 6, to this entry. It also updates the sequence history (Seq-inst.hist, see the Biological Sequences section) of both old and new entries to make a doubly linked list. The GI 5 entry has a pointer that it has been replaced by GI 6, and the GI 6 entry has a pointer showing it replaced GI 5. When NCBI makes a new data release the entry designated GI 6 will be released to represent EMBL entry A00000. However, the ASN.1 form of the data contains an explicit history. A scientist who annotated a coding region on GI 5 can discover that it has been replaced by GI 6. The GI 5 entry can still be retrieved from ID, aligned with GI 6, and the scientist can determine if her annotation is still valid on the new entry. If she annotated using the accession number instead of the GI, of course, she could be out of luck.

Since ID is attempting to order a chaotic world, mistakes will inevitably be made. However, it is clear that in the vast majority of cases it is possible to impose stable ids. As scientists and software begin to use the GI ids and reap the benefits of stable ids, the world may gradually become less chaotic. The Seq-inst.hist data structure can even be used by data suppliers to actively maintain an explicit history without ID having to infer it, which would be the ideal case.

Seq-loc: Locations on a Bioseq

A Seq-loc is a location on a Bioseq of any representation class, nucleic acid or protein. All Bioseqs provide a simple integer coordinate system from 0 to (length -1) and all Seq-locs refer to that coordinate system. All Seq-locs also explicitly the Bioseq (coordinate system) to which they apply with a Seq-id. Most objects which are attached to or reference sequences do so through a Seq-loc. Features are blocks of data attached by a Seq-loc. An alignment is just a collection of correlated Seq-locs. A segmented sequence is built from other sequences by reference to Seq-locs.

Some important methods of the CSeq_loc class and some of the subtype classes (CSeq_interval, CSeq_loc_mix etc.) are:

- CSeq_loc() -- constructors to simplify creation of simple Seq-loc objects.
- Compare() -- compares two Seq-locs if they are defined on the same Bioseq.
- GetTotalRange() -- returns range, covering the whole Seq-loc. If the Seq-loc refers multiple Bioseqs, exception is thrown.
- IsReverseStrand() -- returns true if all ranges in the Seq-loc have reverse strand.
- GetStart(), GetStop() -- return start and stop positions of the Seq-loc. This may be different from GetTotalRange if the related Bioseq is circular or if the order of ranges in the Seq-loc is non-standard.
- GetCircularLength() -- returns length of the Seq-loc. If the sequence length is provided, the method checks whether the Seq-loc is circular and calculates the correct length, even if the location crosses a sequence start.
- CheckId() -- checks whether the Seq-loc refers to only one Seq-id and returns it; otherwise, it sends an exception.
- Add() -- adds a sub-location to the existing one.

Beside these methods, a new class CSeq_loc_CI is defined in Seq_loc.hpp, which provides simplified access to individual ranges of any Seq-loc, regardless of its real type and structure.

null: A Gap

A null Seq-loc can be used in a Seq-loc with many components to indicate a gap of unknown size. For example it is used in segmented sequences to indicate such gaps between the sequenced pieces.

empty: A Gap in an Alignment

A alignment (see Sequence Alignments) may require that every Seq-loc refer to a Bioseq, even for a gap. They empty type fulfills this need.

whole: A Reference to a Whole Bioseq

This is just a shorthand for the Bioseq from 0 to (length -1). This form is falling out of favor at NCBI because it means one must retrieve the referenced Bioseq to determine the length of the location. An interval covering the whole Bioseq is equivalent to this and more useful. On the other hand, if an unstable Seq-id is used here, it always applies to the full length of the Bioseq, even if the length changes. This was the original rationale for this type. And it may still be valid while unstable sequences persist.

int: An Interval on a Bioseq

An interval is a single continuous region of defined length on a Bioseq. A single integer value (Seqinterval.from), another single integer value (Seq-interval.to), and a Seq-id (Seq-interval.id) are required. The "from" and "to" values must be in the range 0 to (length -1) of the Bioseq cited in "id". If there is uncertainty about either the "from" or "to" values, it is expressed in additional fields "fuzz-from" and/or "fuzz-to", and the "from" and "to" values can be considered a "best guess" location. This design means that simple software can ignore fuzzy values, but they are not lost to more sophisticated tools.

The "from" value is ALWAYS less than or equal to the "to" value, no matter what strand the interval is on. It may be convenient for software to present intervals on the minus strand with the "to" value before the "from" value, but internally this is NEVER the case. This requirement means that software which determines overlaps of locations need never treat plus or minus strand locations differently and it greatly simplifies processing.

The value of Seq-interval.strand is the only value different in intervals on the plus or minus strand. Seq-interval.strand is OPTIONAL since it is irrelevant for proteins, but operationally it will DEFAULT to plus strand on nucleic acid locations where it is not supplied.

The plus or minus strand is an attribute on each simple Seq-loc (interval or point) instead of as an operation on an arbitrarily complex location (as in the GenBank/EMBL/DDBJ flatfile Feature Table) since it means even very complex locations can be processed to a base pair location in simple linear order, instead of requiring that the whole expression be processed and resolved first.

packed-int: A Series of Intervals

A Packed-seqint is simply a SEQUENCE OF Seq-interval. That means the location is resolved by evaluating a series of Seq-interval in order. Note that the Seq-intervals in the series need not all be on the same Bioseq or on the same strand.

pnt: A Single Point on a Sequence

A Seq-point is essentially one-half of a Seq-interval and the discussion (above) about fuzziness and strand applies equally to Seq-point.

packed-pnt: A Collection of Points

A Packed-seqpnt is an optimization for attaching a large number of points to a single Bioseq. Information about the Seq-id, strand, or fuzziness need not be duplicated for every point. Of course, this also means it must apply equally to all points as well. This would typically be the case for listing all the cut sites of a certain restriction enzyme, for example.

mix: An Arbitrarily Complex Location

A Seq-loc-mix is simply a SEQUENCE OF Seq-loc. The location is resolved by resolving each Seq-loc in order. The component Seq-locs may be of any complexity themselves, making this definition completely recursive. This means a relatively small amount of software code can process locations of extreme complexity with relative ease.

A Seq-loc-mix might be used to represent a segmented sequence with gaps of unknown length. In this case it would consist of some elements of type "int" for intervals on Bioseqs and some of type "null" representing gaps of unknown length. Another use would be to combine a Seq-interval representing an untranslated leader, with a Packed-seqint from a multi-exon coding region feature, and another Seq-interval representing an untranslated 3' end, to define the extent of an mRNA on a genomic sequence.

equiv: Equivalent Locations

This form is simply a SET OF Seq-locs that are equivalent to each other. Such a construct could be used to represent alternative splicing, for example (and is when translating the GenBank/EMBL/DDBJ location "one-of"). However note that such a location can never resolve to a single result. Further, if there are multiple "equiv" forms in a complex Seq-loc, it is unclear if all possible combinations are valid. In general this construct should be avoided unless there is no alternative.

bond: A Chemical Bond Between Two Residues

The data elements in a Seq-bond are just two Seq-points. The meaning is that these two points have a chemical bond between them (which is different than describing just the location of two points). At NCBI we have restricted its use to covalent bonds. Note that the points may be on the same (intra-chain bond) or different (inter-chain bond) Bioseqs.

feat: A Location Indirectly Referenced Through A Feature

This one is really for the future, when not only Bioseqs, but features have stable ids. The meaning is "the location of this feature". This way one could give a valid location by citing, for example a Gene feature, which would resolve to the location of that gene on a Bioseq. When identifiable features become common (see Sequence Features) this will become a very useful location.

Seq-loc: The C++ Implementation

The following table summarizes the Choice variants for CSeq_loc objects.

Seq-loc

Enum Value	Enum name	ASN.1 name
0	e_not_set	
1	e_Null	null
2	e_Empty	empty
3	e_Whole	whole
4	e_Int	int
5	e_Packed_int	packed-int
6	e_Pnt	pnt

7	e_Packed_pnt	packed-pnt
8	e_Mix	mix
9	e_Equiv	equiv
10	e_Bond	bond
11	e_Feat	feat

Note that e_Mix and e_Equiv Seq-loc types can recursively contain other Seq-locs. Also, the e_Int type (implemented by CSeq_interval) has the following strand enumeration:

Enum Value	**Enum name**	**Notes**
0	e_Na_strand_unknown	
1	e_Na_strand_plus	
2	e_Na_strand_minus	
3	e_Na_strand_both	in forward direction
4	e_Na_strand_both_rev	in reverse direction
5	e_Na_strand_other	

In addition, there are a large number of utility functions for working with SeqLocs described in the chapter on Sequence Utilities. This allow traversal of complex locations, comparison of locations for overlap, conversion of coordinates in locations, and ability to open a window on a Bioseq through a location.

Sequence Features

This section documents data structures used to describe regions of Bioseqs. The types are located in the seqfeat.asn module.

C++ Implementation Notes

In the C++ Toolkit, many types defined in the seqfeat ASN.1 module are extended to simplify access to the feature data. The CSeq_feat class has methods for comparing features by type and location. The CSeqFeatData class defines feature subtypes and qualifiers so that you can better identify individual features.

- Txinit: Transcription Initiation
- Current Genetic Code Table: gc.prt
- ASN.1 Specification: seqfeat.asn

Introduction

A sequence feature (Seq-feat) is a block of structured data (SeqFeatData) explicitly attached to a region of a Bioseq through one or two Seq-locs (see Sequence Locations and Identifiers). The Seq-feat itself can carry information common to all features, as well as serving as the junction between the SeqFeatData and Seq-loc(s). Since a Seq-feat references a Bioseq through an explicit Seq-loc, a Seq-feat is an entity which can stand alone, or be moved between contexts without loss of information. Thus, information ABOUT Bioseqs can be created, exchanged, and compared independently from the Bioseq itself. This is an important attribute of the NCBI data model.

A feature table is a set of Seq-feat gathered together within a Seq-annot (see Biological Sequences). The Seq-annot allows the features to be attributed to a source and be associated with a title or comment. Seq-feats are normally exchanged "packaged" into a feature table.

Seq-feat: Structure of a Feature

A Seq-feat is a data structure common to all features. The fields it contains can be evaluated by software the same way for all features, ignoring the "data" element which is what makes each feature class unique.

id: Features Can Have Identifiers

At this time unique identifiers for features are even less available or controlled than sequence identifiers. However, as molecular biology informatics becomes more sophisticated, it will become not only useful, but essential to be able to cite features as precisely as NCBI is beginning to be able to cite sequences. The Seq-feat.id slot is where these identifiers will go. The Feat-id object for features, meant to be equivalent of the Seq-id object for Bioseqs, is not very fully developed yet. It can accommodate feature ids from the NCBI Backbone database, local ids, and the generic Dbtag type. Look for better characterized global ids to appear here in future as the requirement for structured data exchange becomes increasingly accepted.

data: Structured Data Makes Feature Types Unique

Each type of feature can have a data structure which is specifically designed to accommodate all the requirements of that type with no concern about the requirements of other feature types. Thus a coding region data structure can have fielded elements for reading frame and genetic code, while a tRNA data structure would have information about the amino acid transferred.

This design completely modularizes the components required specifically by each feature type. If a new field is required by a particular feature type, it does not affect any of the others. A new feature type, even a very complex one, can be added without affecting any of the others.

Software can be written in a very modular fashion, reflecting the data design. Functions common to all features (such as determining all features in a sequence region) simply ignore the "data" field and are robust against changes or additions to this component. Functions which process particular types have a well defined data interface unique to each type.

Perhaps a less obvious consequence is code and data reuse. Data objects used in other contexts can be used as features simply by making them a CHOICE in SeqFeatData. For example, the publication feature reuses the Pubdesc type used for Bioseq descriptors. This type includes all the standard bibliographic types (see Bibliographic References) used by MEDLINE or other bibliographic databases. Software which displays, queries, or retrieves publications will work without change on the "data" component of a publication feature because it is EXACTLY THE SAME object. This has profound positive consequences for both data and code development and maintenance.

This modularization also makes it natural to discuss each allowed feature type separately as is done in the SeqFeatData section below.

partial: This Feature is Incomplete

If Seq-feat.partial is TRUE, the feature is incomplete in some (unspecified) way. The details of incompleteness may be specified in more detail in the Seq-feat.location field. This flag allows quick exclusion of incomplete features when doing a database wide survey. It also allows the feature to be flagged when the details of incompleteness may not be know.

Seq-feat.partial should ALWAYS be TRUE if the feature is incomplete, even if Seq-feat.location indicates the incompleteness as well.

except: There is Something Biologically Exceptional

The Seq-feat.except flag is similar to the Seq-feat.partial flag in that it allows a simple warning that there is something unusual about this feature, without attempting to structure a detailed explanation. Again, this allows software scanning features in the database to ignore atypical cases easily. If Seq-feat.except is TRUE, Seq-feat.comment should contain a string explaining the exceptional situation.

Seq-feat.except does not necessarily indicate there is something wrong with the feature, but more that the biological exceeds the current representational capacity of the feature definition and that this may lead to an incorrect interpretation. For example, a coding region feature on genomic DNA where post-transcriptional editing of the RNA occurs would be a biological exception. If one translates the region using the frame and genetic code given in the feature one does not get the protein it points to, but the data supplied in the feature is, in fact, correct. It just does not take into account the RNA editing process.

Ideally, one should try to avoid or minimize exceptions by the way annotation is done. An approach to minimizing the RNA editing problem is described in the "product" section below. If one is forced to use exception consistently, it is a signal that a new or revised feature type is needed.

comment: A Comment About This Feature

No length limit is set on the comment, but practically speaking brief is better.

product: Does This Feature Produce Another Bioseq?

A Seq-feat is unusual in that it can point to two different sequence locations. The "product" location enables two Bioseqs to be linked together in a source/product relationship explicitly. This is very valuable for features which describe a transformation from one Bioseq to another, such as coding region (nucleic acid to protein) or the various RNA types (genomic nucleic acid to RNA product).

This explicit linkage is extremely valuable for connecting diverse types. Linkage of nucleic acid to protein through coding region makes data traversal from gene to product or back simple and explicit, but clearly of profound biological significance. Less obvious, but nonetheless useful is the connection between a tRNA gene and the modified sequence of the tRNA itself, or of a transcribed coding region and an edited mRNA.

Note that such a feature is as valuable in association with its product Bioseq alone as it is with its source Bioseq alone, and could be distributed with either or both.

location: Source Location of This Feature

The Seq-feat.location is the traditional location associated with a feature. While it is possible to use any Seq-loc type in Seq-feat.location, it is recommended to use types which resolve to a single unique sequence. The use of a type like Seq-loc-equiv to represent alternative splicing of exons (similar to the GenBank/EMBL/DDBJ feature table "one-of") is strongly discouraged. Consider the example of such an alternatively spliced coding region. What protein sequence is coded for by such usage? This problem is accentuated by the availability of the "product" slot. Which protein sequence is the product of this coding region? While such a short hand notation may seem attractive at first glance, it is clearly much more useful to represent each splicing alternative, and its associated protein product, times of expression, etc. separately.

qual: GenBank Style Qualifiers

The GenBank/EMBL/DDBJ feature table uses "qualifiers", a combination of a string key and a string value. Many of these qualifiers do not map to the ASN.1 specification, so this provides a means of carrying them in the Seq-feat for features derived from those sources.

title: A User Defined Name

This field is provided for naming features for display. It would be used by end-user software to allow the user to add locally meaningful names to features. This is not an id, as this is provided by the "id" slot.

ext: A User Defined Structured Extension

The "ext" field allows the extension of a standard feature type with a structured User-object (see General Use Objects) defined by a user. For example, a particular scientist may have additional detailed information about coding regions which do not fit into the standard CdRegion data type. Rather than create a completely new feature type, the CdRegion type can be extended by filling in as much of the standard CdRegion fields as possible, then putting the additional information in the User-object. Software which only expects a standard coding region will operate on the extended feature without a problem, while software that can make use of the additional data in the User-object can operate on exactly the same the feature.

cit: Citations For This Feature

This slot is a set of Pubs which are citations about the feature itself, not about the Bioseq as a whole. It can be of any type, although the most common is type "pub", a set of any kind of Pubs. The individual Pubs within the set may be Pub-equivs (see Bibliographic References) to hold equivalent forms for the same publication, so some thought should be given to the process of accessing all the possible levels of information in this seemingly simple field.

exp-ev: Experimental Evidence

If it is known for certain that there is or is not experimental evidence supporting a particular feature, Seq-feat.exp-ev can be "experimental" or "not-experimental" respectively. If the type of evidence supporting the feature is not known, exp-ev should not be given at all.

This field is only a simple flag. It gives no indication of what kind of evidence may be available. A structured field of this type will differ from feature type to feature type, and thus is inappropriate to the generic Seq-feat. Information regarding the quality of the feature can be found in the CdRegion feature and even more detail on methods in the Tx-init feature. Other feature types may gain experimental evidence fields appropriate to their types as it becomes clear what a reasonable classification of that evidence might be.

xref: Linking To Other Features

SeqFeatXrefs are copies of the Seq-feat.data field and (optionally) the Seq-feat.id field from other related features. This is a copy operation and is meant to keep some degree of connectivity or completeness with a Seq-feat that is moved out of context. For example, in a collection of data including a nucleic acid sequence and its translated protein product, there would be a Gene feature on the nucleic acid, a Prot-ref feature on the protein, and a CdRegion feature linking all three together. However, if the CdRegion feature is taken by itself, the name of the translated protein and the name of the gene are not immediately available. The Seq-feat.xref provides a simple way to copy the relevant information. Note that there is a danger to any such copy operation in that the original source of the copied data may be modified without updating the copy. Software should be careful about this, and the best course is to take the original data if it is available to the software, using any copies in xref only as a last resort. If the "id" is included in the xref, this makes it easier for software to keep the copy up to date. But it depends on widespread use of feature ids.

SeqFeatData: Type Specific Feature Data

The "data" slot of a Seq-feat is filled with SeqFeatData, which is just a CHOICE of a variety of specific data structures. They are listed under their CHOICE type below, but for most types a detailed discussion will be found under the type name itself later in this chapter, or in another chapter. That is because most types are data objects in their own right, and may find uses in many other contexts than features.

gene: Location Of A Gene

A gene is a feature of its own, rather than a modifier of other features as in the GenBank/EMBL/DDBJ feature tables. A gene is a heritable region of nucleic acid sequence which confers a measurable phenotype. That phenotype may be achieved by many components of the gene including but not limited to coding regions, promoters, enhancers, terminators, and so on. The gene feature is meant to approximately cover the region of nucleic acid considered by workers in the field to be the gene. This admittedly fuzzy concept has an appealing simplicity and fits in well with higher level views of genes such as genetic maps.

The gene feature is implemented by a Gene-ref object, or a "reference to" a gene. The Gene-ref object is discussed below.

org: Source Organism Of The Bioseq

Normally when a whole Bioseq or set of Bioseqs is from the same organism, the Org-ref (reference to Organism) will be found at the descriptor level of the Bioseq or Bioseq-set (see Biological Sequences). However, in some cases the whole Bioseq may not be from the same organism. This may occur naturally (e.g. a provirus integrated into a host chromosome) or artificially (e.g. recombinant DNA techniques).

The org feature is implemented by an Org-ref object, or a "reference to" an organism. The Orgref is discussed below.

cdregion: Coding Region

A cdregion is a region of nucleic acid which codes for a protein. It can be thought of as "instructions to translate" a nucleic acid, not simply as a series of exons or a reflection of an mRNA or primary transcript. Other features represent those things. Unfortunately, most existing sequences in the database are only annotated for coding region, so transcription and splicing information must be inferred (often inaccurately) from it. We encourage the annotation of transcription features in addition to the coding region. Note that since the cdregion is "instructions to translate", one can represent translational stuttering by having overlapping intervals in the Seq-feat.location. Again, beware of assuming a cdregion definitely reflects transcription.

A cdregion feature is implemented by a Cdregion object, discussed below.

prot: Describing A Protein

A protein feature describes and/or names a protein or region of a protein. It uses a Prot-ref object, or "reference to" a protein, described in detail below.

A single amino acid Bioseq can have many protein features on it. It may have one over its full length describing a pro-peptide, then a shorter one describing the mature peptide. An extreme case might be a viral polyprotein which would have one protein feature for the whole polyprotein, then additional protein features for each of the component mature proteins. One should always take into account the "location" slot of a protein feature.

rna: Describing An RNA

An RNA feature can describe both coding intermediates and structural RNAs using an RNA-ref, or "reference to" an RNA. The RNA-ref is described in more detail below. The Seq-feat.location for an RNA can be attached to either the genomic sequence coding for the RNA, or to the sequence of the RNA itself, when available. The determination of whether the Bioseq the RNA feature is attached to is genomic or an RNA type is made by examining the Bioseq.descr.mol-type, not by making assumptions based on the feature. When both the genomic Bioseq and the RNA Bioseq are both available, one could attach the RNA Seq-feat.location to the genomic sequence and the Seq-feat.product to the RNA to connect them and capture explicitly the process by which the RNA is created.

pub: Publication About A Bioseq Region

When a publication describes a whole Bioseq, it would normally be at the "descr" slot of the Bioseq. However, if it applies to a sub region of the Bioseq, it is convenient to make it a feature. The pub feature uses a Pubdesc (see Biological Sequences for a detailed description) to describe a publication and how it relates to the Bioseq. To indicate a citation about a specific feature (as opposed to about the sequence region in general), use the Seq-feat.cit slot of that feature.

seq: Tracking Original Sequence Sources

The "seq" feature is a simple way to associate a region of sequence with a region of another. For example, if one wished to annotate a region of a recombinant sequence as being from "pBR322 10-50" one would simply use a Seq-loc (see Sequence Locations and Identifiers) for the interval 10-50 on Seq-id pBR322. Software tools could use such information to provide the pBR322 numbering system over that interval.

This feature is really meant to accommodate older or approximate data about the source of a sequence region and is no more than annotation. More specific and computationally useful ways of doing this are (1) create the recombinant sequence as a segmented sequence directly (see Biological Sequences), (2) use the Seq-hist field of a Bioseq to record its history, (3) create alignments (see Sequence Alignments) which are also valid Seq-annots, to indicate more complex relationships of one Bioseq to others.

imp: Importing Features From Other Data Models

The SeqFeatData types explicitly define only certain well understood or widely used feature types. There may be other features contained in databases converted to this specification which are not represented by this ASN.1 specification. At least for GenBank, EMBL, DDBJ, PIR, and SWISS-PROT, these can be mapped to an Imp-feat structure so the features are not lost, although they are still unique to the source database. All these features have the basic form of a string key, a location (carried as the original string), and a descriptor (another string). In the GenBank/EMBL/DDBJ case, any additional qualifiers can be carried on the Seq-feat.qual slot.

GenBank/EMBL/DDBJ use a "location" called "replace" which is actually an editing operation on the sequence which incorporates literal strings. Since the locations defined in this specification are locations on sequences, and not editing operations, features with replace operators are all converted to Imp-feat so that the original location string can be preserved. This same strategy is taken in the face of incorrectly constructed locations encountered in parsing outside databases into ASN.1.

region: A Named Region

The region feature provides a simple way to name a region of a Bioseq (e.g. "globin locus", "LTR", "subrepeat region", etc).

comment: A Comment On A Region Of Sequence

The comment feature allows a comment to be made about any specified region of sequence. Since comment is already a field in Seq-feat, there is no need for an additional type specific data item in this case, so it is just NULL.

bond: A Bond Between Residues

This feature annotates a bond between two residues. A Seq-loc of type "bond" is expected in Seq-feat.location. Certain types of bonds are given in the ENUMERATED type. If the bond type is "other" the Seq-feat.comment slot should be used to explain the type of the bond. Allowed bond types are:

```
disulfide (1) ,
thiolester (2) ,
xlink (3) ,
thioether (4) ,
other (255)
```

site: A Defined Site

The site feature annotates a know site from the following specified list. If the site is "other" then Seq-feat.comment should be used to explain the site.

```
active (1) ,
binding (2) ,
cleavage (3) ,
inhibit (4) ,
modified (5),
glycosylation (6) ,
myristoylation (7) ,
mutagenized (8) ,
metal-binding (9) ,
phosphorylation (10) ,
acetylation (11) ,
amidation (12) ,
methylation (13) ,
hydroxylation (14) ,
sulfatation (15) ,
oxidative-deamination (16) ,
pyrrolidone-carboxylic-acid (17) ,
gamma-carboxyglutamic-acid (18) ,
blocked (19) ,
lipid-binding (20) ,
np-binding (21) ,
dna-binding (22) ,
other (255)
```

rsite: A Restriction Enzyme Cut Site

A restriction map is basically a feature table with rsite features. Software which generates such a feature table could then use any sequence annotation viewer to display its results. Restriction maps generated by physical methods (before sequence is available), can use this feature to create a map type Bioseq representing the ordered restriction map. For efficiency one would probably create one Seq-feat for each restriction enzyme used and used the Packed-pnt Seq-loc in the location slot. See Rsite-ref, below.

user: A User Defined Feature

An end-user can create a feature completely of their own design by using a User-object (see General Use Objects) for SeqFeatData. This provides a means for controlled addition and testing of new feature types, which may or may not become widely accepted or to "graduate" to a defined SeqFeatData type. It is also a means for software to add structured information to Bioseqs for its own use and which may never be intended to become a widely used standard. All the generic feature operations, including display, deletion, determining which features are carried on a sub region of sequence, etc, can be applied to an user feature with no knowledge of the particular User-object structure or meaning. Yet software which recognizes that User-object can take advantage of it.

If an existing feature type is available but lacks certain additional fields necessary for a special task or view of information, then it should be extended with the Seq-feat.ext slot, rather than building a complete user feature de novo.

txinit: Transcription Initiation

This feature is used to designate the region of transcription initiation, about which considerable knowledge is available. See Txinit, below.

num: Applying Custom Numbering To A Region

A Numbering object can be used as a Bioseq descriptor to associate various numbering systems with an entire Bioseq. When used as a feature, the numbering system applies only to the region in Seq-feat.location. This make multiple, discontinuous numbering systems available on the same Bioseq. See Biological Sequences for a description of Numbering, and also Seq-feat.seq, above, for an alternative way of applying a sequence name and its numbering system to a sequence region.

psec-str: Protein Secondary Structure

Secondary structure can be annotated on a protein sequence using this type. It can be predicted by algorithm (in which case Seq-feat.exp-ev should be "not-experimental") or by analysis of the known protein structure (Seq-feat.exp-ev = "experimental"). Only three types of secondary structure are currently supported. A "helix" is any helix, a "sheet" is beta sheet, and "turn" is a beta or gamma turn. Given the controversial nature of secondary structure classification (not be mention prediction), we opted to keep it simple until it was clear that more detail was really necessary or understood.

non-std-residue: Unusual Residues

When an unusual residue does not have a direct sequence code, the "best" standard substitute can be used in the sequence and the residue can be labeled with its real name. No attempt is made to enforce a standard nomenclature for this string.

het: Heterogen

In the PDB structural database, non-biopolymer atoms associated with a Bioseq are referred to as "heterogens". When a heterogen appears as a feature, it is assumed to be bonded to the sequence positions in Seq-feat.location. If there is no specific bonding information, the heterogen will appear as a descriptor of the Bioseq. The Seq-loc for the Seq-feat.location will probably be a point or points, not a bond. A Seq-loc of type bond is between sequence residues.

Seq-feat Implementation in C++

The C++ implementation of a Seq-feat is mostly straightforward. However, some explanation of the "id" and "data" slots will be helpful. Both are implemented as a Choice and contained in the CSeq_feat object. The tables below summarize the id and data choice variants.

SeqFeat.id

ASN.1 name	Value
(not present)	0
gibb	1
giim	2
local	3
general	4

SeqFeat.data

ASN.1 name	Value
(not present)	0
gene	1
org	2
cdregion	3

prot	4
rna	5
pub	6
seq	7
imp	8
region	9
comment	10
bond	11
site	12
rsite	13
user	14
txinit	15
num	16
psec-str	17
non-std-residue	18
het	19
biosrc	20
clone	21

Of course, within the software tools for producing GenBank, report, or other formats from ASN.1 are functions to format and display features as well. There are some functions to manipulate the CSeqFeatData objects, such as the translation of a CdRegion, and a host of functions to use and compare the Seq-locs of "product" and "location" or easily access and use the sequence regions they point to. These functions are discussed in the Sequence Utilities chapter. Additional functions, described in Exploring The Data, allow one to easily locate features of interest by type, in arbitrarily complex objects.

CdRegion: Coding Region

A CdRegion, in association with a Seq-feat, is considered "instructions to translate" to protein. The Seq-locs used by the Seq-feat do not necessarily reflect the exon structure of the primary transcript (although they often do). A Seq-feat of type CdRegion can point both to the source nucleic acid and to the protein sequence it produces. Most of the information about the source nucleic acid (such as the gene) or the destination protein (such as its name) is associated directly with those Bioseqs. The CdRegion only serves as a link between them, and as a method for explicitly encoding the information needed to derive one from the other.

orf: Open Reading Frame

CdRegion.orf is TRUE if the coding region is only known to be an open reading frame. This is a signal that nothing is known about the protein product, or even if it is produced. In this case the translated protein sequence will be attached, but there will be no other information associated with it. This flag allows such very speculative coding regions to be easily ignored when scanning the database for genuine protein coding regions.

The orf flag is not set when any reasonable argument can be made that the CdRegion is really expressed, such as detection of mRNA or strong sequence similarity to known proteins.

Translation Information

CdRegion has several explicit fields to define how to translate the coding region. Reading frame is explicitly given or defaults to frame one.

The genetic code is assumed to be the universal code unless given explicitly. The code itself is given, rather than requiring software to determine the code at run-time by analyzing the phylogenetic position of the Bioseq. Genetic code is described below.

Occasionally the genetic code is not followed at specific positions in the sequence. Examples are the use of alternate initiation codons only in the first position, the effects of suppresser tRNAs, or the addition of selenocysteine. The Code-break object specifies the three bases of the codon in the Bioseq which is treated differently and the amino acid which is generated at that position. During translation the genetic code is followed except at positions indicated by Code-breaks, where the instructions in the Code-break are followed instead.

Problems With Translations

In a surprising number of cases an author publishes both a nucleic acid sequence and the protein sequence produced by its coding region, but the translation of the coding region does not yield the published protein sequence. On the basis of the publication it is not possible to know for certain which sequence is correct. In the NCBI Backbone database both sequences are preserved as published by the author, but the conflict flag is set to TRUE in the CdRegion. If available, the number of gaps and mismatches in the alignment of the translated sequence to the published protein sequence are also given so a judgment can be made about the severity of the problem.

Genetic Codes

A Genetic-code is a SET which may include one or more of a name, integer id, or 64 cell arrays of amino acid codes in different alphabets. Thus, in a CdRegion, one can either refer to a genetic code by name or id; provide the genetic code itself, or both. Tables of genetic codes are provided in the NCBI software release with most possibilities filled in.

The Genetic-code.name is a descriptive name for the genetic code, mainly for display to humans. The integer id refers to the ids in the gc.val (binary ASN.1) or gc.prt (text ASN.1) file of genetic codes maintained by NCBI, distributed with the software tools and Entrez releases, and published in the GenBank/EMBL/DDBJ feature table document. Genetic-code.id is the best way to explicitly refer to a genetic code.

The genetic codes themselves are arrays of 64 amino acid codes. The index to the position in the array of the amino acid is derived from the codon by the following method:

index = (base1 * 16) + (base2 * 4) + base3

where T=0, C=1, A=2, G=3

Note that this encoding of the bases is not the same as any of the standard nucleic acid encoding described in Biological Sequence. This set of values was chosen specifically for genetic codes because it results in the convenient groupings of amino acid by codon preferred for display of genetic code tables.

The genetic code arrays have names which indicate the amino acid alphabet used (e.g. ncbieaa). The same encoding technique is used to specify start codons. Alphabet names are prefixed with "s" (e.g. sncbieaa) to indicate start codon arrays. Each cell of a start codon array contains either the gap code ("-" for ncbieaa) or an amino acid code if it is valid to use the codon as a start codon. Currently all starts are set to code for methionine, since it has never been convincingly demonstrated that a protein can start with any other amino acid. However, if other amino acids are shown to be used as starts, this structure can easily accommodate that information.

The contents of gc.prt, the current supported genetic codes, is given at the end of this chapter.

C++ Implementation Of Genetic Codes

The GeneticCode type is summarized as follows:

GeneticCode Elements

ASN.1 name	Value
name	1
id	2
ncbieaa	3
ncbi8aa	4
ncbistdaa	5
sncbieaa	6
sncbi8aa	7
sncbistdaa	8

Rsite-ref: Reference To A Restriction Enzyme

This simple data structure just references a restriction enzyme. It is a choice of a simple string (which may or may not be from a controlled vocabulary) or a Dbtag, in order to cite an enzyme from a specific database such as RSITE. The Dbtag is preferred, if available.

Note that this reference is not an Rsite-entry which might contain a host of information about the restriction enzyme, but is only a reference to the enzyme.

RNA-ref: Reference To An RNA

An RNA-ref allows naming and a minimal description of various RNAs. The "type" is a controlled vocabulary for dividing RNAs into broad, well accepted classes. The "pseudo" field is used for RNA pseudogenes.

The "ext" field allows the addition of structure information appropriate to a specific RNA class as appropriate. The "name" extension allows naming the "other" type or adding a modifier, such as "28S" to rRNA. For tRNA there is a structured extension which as fields for the amino acid transferred, drawn from the standard amino acid alphabets, and a value for one or more codons that this tRNA recognizes. The values of the codons are calculated as a number from 0 to 63 using the same formula as for calculating the index to Genetic Codes, above.

As nomenclature and attributes for classes of RNAs becomes better understood and accepted, the RNA-ref.ext will gain additional extensions.

Gene-ref: Reference To A Gene

A Gene-ref is not intended to carry all the information one might want to know about a gene, but to provide a small set of information and reference some larger body of information, such as an entry in a genetic database.

The "locus" field is for the gene symbol, preferably an official one (e.g. "Adh"). The "allele" field is for an allele symbol (e.g. "S"). The "desc" field is for a descriptive name for the gene (e.g. "Alcohol dehydrogenase, SLOW allele"). One should fill in as many of these fields as possible.

The "maploc" field accepts a string with a map location using whatever conventions are appropriate to the organism. This field is hardly definitive and if up to date mapping information is desired a true mapping database should always be consulted.

If "pseudo" is TRUE, this is a pseudogene.

The "db" field allows the Gene-ref to be attached to controlled identifiers from established gene databases. This allows a direct key to a database where gene information will be kept up to date without requiring that the rest of the information in the Gene-ref necessarily be up to date as well. This type of foreign key is essential to keeping loosely connected data up to date and NCBI is encouraging gene databases to make such controlled keys publicly available.

The "syn" field holds synonyms for the gene. It does not attempt to discriminate symbols, alleles, or descriptions.

Prot-ref: Reference To A Protein

A Prot-ref is meant to reference a protein very analogous to the way a Gene-ref references a gene. The "name" field is a SET OF strings to allow synonyms. The first name is presumed to be the preferred name by software tools. Since there is no controlled vocabulary for protein names this is the best that can be done at this time. "ADH" and "alcohol dehydrogenase" are both protein names.

The "desc" field is for a description of the protein. This field is often not necessary if the name field is filled in, but may be informative in some cases and essential in cases where the protein has not yet been named (e.g. ORF21 putative protein).

The "ec" field contains a SET of EC numbers. These strings are expected to be only numbers separated by periods (no leading "EC"). Sometimes the last few positions will be occupied by dashes or not filled in at all if the protein has not been fully characterized. Examples of EC numbers are (1.14.13.8 or 1.14.14.- or 1.14.14.3 or 1.14.--.-- or 1.14).

The "activity" field allows the various known activities of the protein to be specified. This can be very helpful, especially when the name is not informative.

The "db" field is to accommodate keys from protein databases. While protein nomenclature is not well controlled, there are subfields such as immunology which have controlled names. There are also databases which characterize proteins in other ways than sequence, such as 2-d spot databases which could provide such a key.

Txinit: Transcription Initiation

This is an example of a SeqFeatData block designed and built by a domain expert, an approach the NCBI strongly encourages and supports. The Txinit structure was developed by Philip Bucher and David Ghosh. It carries most of the information about transcription initiation represented in the Eukaryotic Promoter Database (EPD). The Txinit structure carries a host of detailed experimental information, far beyond the simple "promoter" features in GenBank/EMBL/DDBJ. EPD is released as a database in its own right and as Txinit Seq-feats. NCBI will be incorporating the EPD in its feature table form to provide expert annotation of the sequence databases in the manner described in the Data Model chapter.

The Txinit object is well described by its comments in the ASN.1 definition. The best source of more in depth discussion of these fields is in the EPD documentation, and so it will not be reproduced here.

Current Genetic Code Table: gc.prt

```
--**************************************************************************
-- This is the NCBI genetic code table
-- Base 1-3 of each codon have been added as comments to facilitate
-- readability at the suggestion of Peter Rice, EMBL
--*************************************************************************
Genetic-code-table ::= {
{
name "Standard" ,
name "SGC0" ,
id 1 ,
ncbieaa "FFLLSSSSYY**CC*WLLLLPPPPHHQQRRRRIIIMTTTTNNKKSSRRVVVVAAAADDEEGGGG",
sncbieaa "-----------------------------------M----------------------------"
-- Base1 TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
-- Base2 TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
-- Base3 TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
} ,
{
name "Vertebrate Mitochondrial" ,
name "SGC1" ,
id 2 ,
ncbieaa "FFLLSSSSYY**CCWWLLLLPPPPHHQQRRRRIIMMTTTTNNKKSS**VVVVAAAADDEEGGGG",
sncbieaa "--------------------------------MMMM---------------M------------"
-- Base1 TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
-- Base2 TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
-- Base3 TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
} ,
{
name "Yeast Mitochondrial" ,
name "SGC2" ,
id 3 ,
ncbieaa "FFLLSSSSYY**CCWWTTTTPPPPHHQQRRRRIIMMTTTTNNKKSSRRVVVVAAAADDEEGGGG",
sncbieaa "-----------------------------------M----------------------------"
-- Base1 TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
-- Base2 TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
-- Base3 TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
} ,
{
name "Mold Mitochondrial and Mycoplasma" ,
name "SGC3" ,
id 4 ,
ncbieaa "FFLLSSSSYY**CCWWLLLLPPPPHHQQRRRRIIIMTTTTNNKKSSRRVVVVAAAADDEEGGGG",
sncbieaa "-----------------------------------M----------------------------"
-- Base1 TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
-- Base2 TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
-- Base3 TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
} ,
{
name "Invertebrate Mitochondrial" ,
name "SGC4" ,
id 5 ,
ncbieaa "FFLLSSSSYY**CCWWLLLLPPPPHHQQRRRRIIMMTTTTNNKKSSSSVVVVAAAADDEEGGGG",
sncbieaa "---M----------------------------M-MM----------------------------"
-- Base1 TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
-- Base2 TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
-- Base3 TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
} ,
{
name "Ciliate Macronuclear and Daycladacean" ,
name "SGC5" ,
id 6 ,
ncbieaa "FFLLSSSSYYQQCC*WLLLLPPPPHHQQRRRRIIIMTTTTNNKKSSRRVVVVAAAADDEEGGGG",
sncbieaa "-----------------------------------M----------------------------"
-- Base1 TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
-- Base2 TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
-- Base3 TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
} ,
{
name "Protozoan Mitochondrial (and Kinetoplast)" ,
name "SGC6" ,
id 7 ,
ncbieaa "FFLLSSSSYY**CCWWLLLLPPPPHHQQRRRRIIIMTTTTNNKKSSRRVVVVAAAADDEEGGGG",
sncbieaa "--MM---------------M------------MMMM---------------M------------"
-- Base1 TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
-- Base2 TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
-- Base3 TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
} ,
{
name "Plant Mitochondrial/Chloroplast (posttranscriptional variant)" ,
name "SGC7" ,
id 8 ,
ncbieaa "FFLLSSSSYY**CC*WLLLLPPPPHHQQRRRWIIIMTTTTNNKKSSRRVVVVAAAADDEEGGGG",
sncbieaa "--M-----------------------------MMMM---------------M------------"
-- Base1 TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
-- Base2 TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
-- Base3 TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
} ,
{
name "Echinoderm Mitochondrial" ,
name "SGC8" ,
id 9 ,
ncbieaa "FFLLSSSSYY**CCWWLLLLPPPPHHQQRRRRIIIMTTTTNNNKSSSSVVVVAAAADDEEGGGG",
sncbieaa "-----------------------------------M----------------------------"
-- Base1 TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
-- Base2 TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
-- Base3 TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
} ,
{
name "Euplotid Macronuclear" ,
name "SGC9" ,
id 10 ,
ncbieaa "FFLLSSSSYY*QCCCWLLLLPPPPHHQQRRRRIIIMTTTTNNKKSSRRVVVVAAAADDEEGGGG",
sncbieaa "-----------------------------------M----------------------------"
-- Base1 TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
-- Base2 TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
-- Base3 TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
} ,
{
name "Eubacterial" ,
id 11 ,
ncbieaa "FFLLSSSSYY**CC*WLLLLPPPPHHQQRRRRIIIMTTTTNNKKSSRRVVVVAAAADDEEGGGG",
sncbieaa "---M---------------M------------M--M---------------M------------"
-- Base1 TTTTTTTTTTTTTTTTCCCCCCCCCCCCCCCCAAAAAAAAAAAAAAAAGGGGGGGGGGGGGGGG
-- Base2 TTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGGTTTTCCCCAAAAGGGG
-- Base3 TCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAGTCAG
}
}
```

Sequence Alignments

Sequence Alignments

Introduction

A sequence alignment is a mapping of the coordinates of one Bioseq onto the coordinates of one or more other Bioseqs. Such a mapping may be associated with a score and/or a method for doing the alignment. An alignment can be generated algorithmically by software or manually by a scientist. The Seq-align object is designed to capture the final result of the process, not the process itself.

A Seq-align is one of the forms of Seq-annot and is as acceptable a sequence annotation as a feature table. Seq-aligns would normally be "packaged" in a Seq-annot for exchange with other tools or databases so the alignments can be identified and given a title.

The most common sequence alignment is from one sequence to another with a one to one relationship between the aligned residues of one sequence with the residues of the other (with allowance for gaps). Two types of Seq-align types, Dense-seg and Dense-diag are specifically for this type of alignment. The Std-seg, on the other hand, is very generic and does not assume that the length of one aligned region is necessarily the same as the other. This permits expansion and contraction of one Bioseq relative to another, which is necessary in the case of a physical map Bioseq aligned to a genetic map Bioseq, or a sequence Bioseq aligned with any map Bioseq.

All the forms of Seq-align are composed of segments. Each segment is an aligned region which contains only sequence or only a gap for any sequence in the alignment. Below is a three dimensional alignment with six segments:

```
Seq-ids
id=100 AAGGCCTTTTAGAGATGATGATGATGATGA
id=200 AAGGCCTaTTAG.......GATGATGATGA
id=300 ....CCTTTTAGAGATGATGAT....ATGA
| 1 | 2 | 3 |4| 5 | 6| Segments
```

Taking only two of the sequences in a two way alignment, only three segments are needed to define the alignment:

```
Seq-ids
id=100 AAGGCCTTTTAGAGATGATGATGATGATGA
id=200 AAGGCCTaTTAG.......GATGATGATGA
| 1 | 2 | 3 | Segments
```

Seq-align

A Seq-align is a collection of segments representing one complete alignment. The whole Seq-align may have a Score representing some measure of quality or attributing the method used to build the Seq-align. In addition, each segment may have a score for that segment alone.

type: global

A global alignment is the alignment of Bioseqs over their complete length. It expresses the relationship between the intact Bioseqs. As such it is typically used in studies of homology between closely related proteins or genomes where there is reason to believe they share a common origin over their complete lengths.

The segments making up a global alignment are assumed to be connected in order from first to last to make up the alignment, and that the full lengths of all sequences will be accounted for in the alignment.

type: partial

A partial alignment only defines a relationship between sequences for the lengths actually included in the alignment. No claim is made that the relationship pertains to the full lengths of any of the sequences.

Like a global alignment, the segments making up a partial alignment are assumed to be connected in order from first to last to make up the alignment. Unlike a global alignment, it is not assumed the alignment will necessarily account for the full lengths of any or all sequences.

A partial or global alignment may use either the denseg choice of segment (for aligned Bioseqs with one to one residue mappings, such as protein or nucleic acid sequences) or the std choice for any Bioseqs including maps. In both cases there is an ordered relationship between one segment and the next to make the complete alignment.

type: diags

A Seq-align of type diags means that each segment is independent of the next and no claims are made about the reasonableness of connecting one segment to another. This is the kind of relationship shown by a "dot matrix" display. A series of diagonal lines in a square matrix indicate unbroken regions of similarity between the sequences. However, diagonals may overlap multiple times, or regions of the matrix may have no diagonals at all. The diags type of alignment captures that kind of relationship, although it is not limited to two dimensions as a dot matrix is.

The diags type of Seq-align may use either the dendiag choice of segment (for aligned Bioseqs with one to one residue mappings, such as protein or nucleic acid sequences) or the std choice for any Bioseqs including maps. In both cases the SEQUENCE OF does not imply any ordered relationship between one segment and the next. Each segment is independent of any other.

Type:disc

A discontinuous alignment is a set of alignments between two or more sequences. The alignments in the set represent the aligned chunks, broken by unaligned regions (represented by the implicit gaps in-between the alignments in the set).

Each chunk is a non-recursive Seq-align of type "global" or "partial" and with the same dimension. Seq-ids in all Seq-aligns are identical (and in the same order).

Examples of usage include mRNA-to-genomic alignments representing exons or genomic-to-genomic alignments containing unaligned regions.

dim: Dimensionality Of The Alignment

Most scientists are familiar with pairwise, or two dimensional, sequence alignments. However, it is often useful to align sequences in more dimensions. The dim attribute of Seq-align indicates the number of sequences which are **simultaneously** aligned. A three dimensional alignment is a true three way alignment (ABC), not three pairwise alignments (AB, AC, BC). Three pairwise alignments are three Seq-align objects, each with dimension equal to two.

Another common situation is when many sequences are aligned to one, as is the case of a merge of a number of components into a larger sequence, or the relationship of many mutant alleles to the wild type sequence. This is also a collection of two dimensional alignments, where one of the Bioseqs is common to all alignments. If the wild type Bioseq is A, and the mutants are B, C, D, then the Seq-annot would contain three two dimensional alignments, AB, AC, AD.

The dim attribute at the level of the Seq-align is **optional**, while the dim attribute is required on **each** segment. This is because it is convenient for a global or partial alignment to know the dimensionality for the whole alignment. It is also an integrity check that every segment in such a Seq-align has the same dimension. For diags however, the segments are independent of each other, and may even have different dimensions. This would be true for algorithms that locate the best n-way diagonals, where n can be 2 to the number of sequences. For a simple dot-matrix, all segments would be dimension two.

Score: Score Of An Alignment Or Segment

A Score contains an id (of type Object-id) which is meant to identify the method used to generate the score. It could be a string (e.g. "BLAST raw score", "BLAST p value") or an integer for use by a software system planning to process a number of defined values. The value of the Score is either an integer or real number. Both Seq-align and segment types allow more than one Score so that a variety of measures for the same alignment can be accommodated.

Dense-diag: Segments For diags Seq-align

A Seq-align of type diags represents a series of unconnected diagonals as a SEQUENCE OF Dense-diag. Since each Dense-diag is unrelated to the next the SEQUENCE OF just suggests a presentation order. It does not imply anything about the reasonableness of joining one Dense-diag to the next. In fact, for a multi-sequence comparison, each Dense-diag may have a different dimension and/or include Bioseqs not included by another Dense-diag.

A single Dense-diag defines its dimension with dim. There should be dim number of Seq-id in ids, indicating the Bioseqs involved in the segment, in order. There should be dim number of integers in starts (offsets into the Bioseqs, starting with 0, as in any Seq-loc) indicating the first (lowest numbered) residue of each Bioseq involved in the segment is, in the same order as ids. The len indicates the length of all Bioseqs in the segment. Thus the last residue involved in the segment for every Bioseq will be its start plus len - 1.

In the case of nucleic acids, if any or all of the segments are on the complement strand of the original Bioseq, then there should be dim number of Na-strand in len in the same order as ids, indicating which segments are on the plus or minus strands. The fact that a segment is on the minus strand or not does NOT affect the values chosen for starts. It is still the lowest numbered offset of a residue involved in the segment.

Clearly all Bioseq regions involved in a Dense-diag must have the same length, so this form does not allow stretching of one Bioseq compared to another, as may occur when comparing a genetic map Bioseq to a physical map or sequence Bioseq. In this case one would use Std-seg.

Dense-seg: Segments for "global" or "partial" Seq-align

A Dense-seg is a single entity which describes a complete global or partial alignment containing many segments. Like Dense-diag above, it is only appropriate when there is no stretching of the Bioseq coordinates relative to each other (as may happen when aligning a physical to a genetic map Bioseq). In that case, one would use a SEQUENCE OF Std-seg, described below.

A Dense-seg must give the dimension of the alignment in dim and the number of segments in the alignment in numseg. The ids slot must contain dim number of Seq-ids for the Bioseqs used in the alignment.

The starts slot contains the lowest numbered residue contained in each segment, in ids order. The starts slot should have numseg times dim integers, or the start of each Bioseq in the first segment in ids order, followed by the start of each Bioseq in the second segment in ids order and so on. A start of minus one indicates that the Bioseq is not present in the segment (i.e. a gap in a Bioseq).

The lens slot contains the length of each segment in segment order, so lens will contain numseg integers.

If any or all of the sequences are on the minus strand of the original Bioseq, then there should be numseg times dim Na-strand values in len in the same order as starts. Whether a sequence segment is on the plus or minus strand has **no** effect on the value selected for starts. It is **always** the lowest numbered residue included in the segment.

The scores is a SEQUENCE OF Score, one for each segment. So there should be numseg Scores, if scores is filled. A single Score for the whole alignment would appear in the score slot of the Seq-align.

The three dimensional alignment show above is repeated below, followed by its ASN.1 encoding into a Seq-align using Dense-seg. The Seq-ids are given in the ASN.1 as type "local".

```
 Seq-ids

 id=100 AAGGCCTTTTAGAGATGATGATGATGATGA
 id=200 AAGGCCTaTTAG.......GATGATGATGA
 id=300 ....CCTTTTAGAGATGATGAT....ATGA
 | 1 | 2 | 3 |4| 5 | 6| Segments

Seq-align ::= {
 type global ,
 dim 3 ,
 segs denseg {
 dim 3 ,
 numseg 6 ,
 ids {
 local id 100 ,
 local id 200 ,
 local id 300 } ,
 starts { 0,0,-1, 4,4,0, 12,-1,8, 19,12,15, 22,15,-1, 26,19,18 } ,
 lens { 4, 8, 7, 3, 4, 4 } } }
```

Std-seg: Aligning Any Bioseq Type With Any Other

A SEQUENCE OF Std-seg can be used to describe any Seq-align type on any types of Bioseqs. A Std-seg is very purely a collection of correlated Seq-locs. There is no requirement that the length of each Bioseq in a segment be the same as the other members of the segment or that the same Seq-loc type be used for each member of the segment. This allows stretching of one Bioseq relative to the other(s) and potentially very complex descriptions of relationships between sequences.

Each Std-seg must give its dimension, so it can be used for diags. Optionally it can give the Seq-ids for the Bioseqs used in the segment (again a convenience for Seq-align of type diags). The loc slot gives the locations on the Bioseqs used in this segment. As usual, there is also a place for various Score(s) associated with the segment. The example given above is presented again, this time as a Seq-align using Std-segs. Note the use of Seq-loc type "empty" to indicate a gap. Alternatively one could simply change the dim for each segment to exclude the Bioseqs not present in the segment, although this would require more interpretation by software.

```
 Seq-ids
 id=100 AAGGCCTTTTAGAGATGATGATGATGATGA
 id=200 AAGGCCTaTTAG.......GATGATGATGA
 id=300 ....CCTTTTAGAGATGATGAT....ATGA
 | 1 | 2 | 3 |4| 5 | 6| Segments
```

```
Seq-align ::= {
 type global ,
 dim 3 ,
 segs std {
 {
 dim 3 ,
 loc {
 int {
 id local id 100 ,
 from 0 ,
 to 3
 } ,
 int {
 id local id 200 ,
 from 0 ,
 to 3
 } ,
 empty local id 300
 }
 } ,
 {
 dim 3 ,
 loc {
 int {
 id local id 100 ,
 from 4 ,
 to 11
 } ,
 int {
 id local id 200 ,
 from 4 ,
 to 11
 } ,
 int {
 id local id 300 ,
 from 0 ,
 to 7
 }
 }
 } ,
 {
 dim 3 ,
 loc {
 int {
 id local id 100 ,
 from 12 ,
 to 18
 } ,
 empty local id 200 ,
 int {
 id local id 300 ,
 from 8 ,
 to 14
 }
 }
 } ,
 {
 dim 3 ,
 loc {
 int {
 id local id 100 ,
 from 19 ,
 to 21
 } ,
 int {
 id local id 200 ,
 from 12 ,
 to 14
 } ,
 int {
 id local id 300 ,
 from 15 ,
 to 17
 }
 }
 } ,
 {
 dim 3 ,
 loc {
 int {
 id local id 100 ,
 from 22 ,
 to 25
 } ,
 int {
 id local id 200 ,
 from 15 ,
 to 18
 } ,
 empty local id 300
 }
 } ,
 {
 dim 3 ,
 loc {
 int {
 id local id 100 ,
 from 26 ,
 to 29
 } ,
 int {
 id local id 200 ,
 from 19 ,
 to 22
 } ,
 int {
 id local id 300 ,
 from 18 ,
 to 21
 }
 }
 }
 }
}
```

Clearly the Std-seg method should only be used when its flexibility is required. Nonetheless, there is no ready substitute for Std-seg when flexibility is demanded.

C++ Implementation Notes

The C++ Toolkit adds several methods to the classes generated from ASN.1 specifications to simplify alignment data access and manipulation. The CSeq_align class has methods returning Seq-id, start, stop, and strand for a particular alignment row, regardless of its representation; it allows swapping alignment rows or converting the alignment from one type to another. The CDense_seg class extends the default set of alignment members with sequence character width (1 or 3, depending on molecule type).

Sequence Graphs

The Sequence Graphs section describes the Seq-graph type used to associate some analytical data with a region on a Bioseq. The type definition is located in the seqres.asn module.

- Introduction
- Seq-graph: Graph on a Bioseq
- ASN.1 Specification: seqres.asn

Introduction

Analytical tools can attach results to Bioseqs in named collections as Seq-annots. This allows analytical programs developed from various sources to add information to a standard object (the Bioseq) and then let a single program designed for displaying a Bioseq and its associated information show the analytical results in an integrated fashion. Feature tables have been discussed previously, and can serve as the vehicle for results from restriction mapping programs, motif searching programs, open reading frame locators, and so on. Alignment programs and curator tools can produce Seq-annots containing Seq-aligns. In this chapter we present the third annotation type, a graph, which can be used to show properties like G+C content, surface potential, hydrophobicity, and so on.

Seq-graph: Graph on a Bioseq

A Seq-graph defines some continuous set of values over a defined interval on a Bioseq. It has slots for a title and a comment. The "loc" field defines the region of the Bioseq to which the graph applies. Titles can be given for the X (graph value) axis and/or the Y (sequence axis) of the graph. The "comp" slot allows a compression to supplied (i.e. how many residues are represented by a single value of the graph?). Compression is assumed to be one otherwise. Scaling values, a and b can be used to scale the values given in the Seq-graph to those displayed on the graph (by the formula "display value" = (a times "graph value") plus b). Finally, the number of values in the graph must be given (and should agree with the length of "loc" divided by "comp").

The graphs themselves can be coded as byte, integer, or real values. Each type defines the maximum and minimum values to show on the graph (no given values need necessarily reach the maximum or minimum) and the value along which to draw the X axis of the graph.

Common ASN.1 Specifications

Following are the ASN.1 specifications referenced in this chapter:

- general.asn
- biblio.asn
- pub.asn
- medline.asn
- seq.asn
- seqblock.asn
- seqcode.asn
- seqset.asn
- seqloc.asn
- seqfeat.asn
- seqalign.asn
- seqres.asn

ASN.1 Specification: general.asn

See also the online-version of this specification, which may be more up-to-date.

```
--$Revision$
--**********************************************************************
--
-- NCBI General Data elements
-- by James Ostell, 1990
-- Version 3.0 - June 1994
--
--**********************************************************************

NCBI-General DEFINITIONS ::=
BEGIN

EXPORTS Date, Person-id, Object-id, Dbtag, Int-fuzz, User-object, User-field;

-- StringStore is really a VisibleString. It is used to define very
-- long strings which may need to be stored by the receiving program
-- in special structures, such as a ByteStore, but it's just a hint.
-- AsnTool stores StringStores in ByteStore structures.
-- OCTET STRINGs are also stored in ByteStores by AsnTool
--
-- typedef struct bsunit { /* for building multiline strings */
 -- Nlm_Handle str; /* the string piece */
 -- Nlm_Int2 len_avail,
 -- len;
 -- struct bsunit PNTR next; } /* the next one */
-- Nlm_BSUnit, PNTR Nlm_BSUnitPtr;
--
-- typedef struct bytestore {
 -- Nlm_Int4 seekptr, /* current position */
 -- totlen, /* total stored data length in bytes */
 -- chain_offset; /* offset in ByteStore of first byte in curchain */
 -- Nlm_BSUnitPtr chain, /* chain of elements */
 -- curchain; /* the BSUnit containing seekptr */
-- } Nlm_ByteStore, PNTR Nlm_ByteStorePtr;
--
-- AsnTool incorporates this as a primitive type, so the definition
-- is here just for completeness
--
-- StringStore ::= [APPLICATION 1] IMPLICIT OCTET STRING
--

-- BigInt is really an INTEGER. It is used to warn the receiving code to
expect
-- a value bigger than Int4 (actually Int8). It will be stored in
DataVal.bigintvalue
--
-- Like StringStore, AsnTool incorporates it as a primitive. The definition
would be:
-- BigInt ::= [APPLICATION 2] IMPLICIT INTEGER
--

-- Date is used to replace the (overly complex) UTCTtime, GeneralizedTime
-- of ASN.1
-- It stores only a date
--

Date ::= CHOICE {
 str VisibleString , -- for those unparsed dates
 std Date-std } -- use this if you can

Date-std ::= SEQUENCE { -- NOTE: this is NOT a unix tm struct
 year INTEGER , -- full year (including 1900)
 month INTEGER OPTIONAL , -- month (1-12)
 day INTEGER OPTIONAL , -- day of month (1-31)
 season VisibleString OPTIONAL , -- for "spring", "may-june", etc
 hour INTEGER OPTIONAL , -- hour of day (0-23)
 minute INTEGER OPTIONAL , -- minute of hour (0-59)
 second INTEGER OPTIONAL } -- second of minute (0-59)

-- Dbtag is generalized for tagging
-- eg. { "Social Security", str "023-79-8841" }
-- or { "member", id 8882224 }
```

```
Dbtag ::= SEQUENCE {
 db VisibleString , -- name of database or system
 tag Object-id } -- appropriate tag

-- Object-id can tag or name anything
--

Object-id ::= CHOICE {
 id INTEGER ,
 str VisibleString }

-- Person-id is to define a std element for people
--

Person-id ::= CHOICE {
 dbtag Dbtag , -- any defined database tag
 name Name-std , -- structured name
 ml VisibleString , -- MEDLINE name (semi-structured)
 -- eg. "Jones RM"
 str VisibleString, -- unstructured name
 consortium VisibleString } -- consortium name

Name-std ::= SEQUENCE { -- Structured names
 last VisibleString ,
 first VisibleString OPTIONAL ,
 middle VisibleString OPTIONAL ,
 full VisibleString OPTIONAL , -- full name eg. "J. John Smith, Esq"
 initials VisibleString OPTIONAL, -- first + middle initials
 suffix VisibleString OPTIONAL , -- Jr, Sr, III
 title VisibleString OPTIONAL } -- Dr., Sister, etc

--**** Int-fuzz **********************************************
--*
--* uncertainties in integer values

Int-fuzz ::= CHOICE {
 p-m INTEGER , -- plus or minus fixed amount
 range SEQUENCE { -- max to min
 max INTEGER ,
 min INTEGER } ,
 pct INTEGER , -- % plus or minus (x10) 0-1000
 lim ENUMERATED { -- some limit value
 unk (0) , -- unknown
 gt (1) , -- greater than
 lt (2) , -- less than
 tr (3) , -- space to right of position
 tl (4) , -- space to left of position
 circle (5) , -- artificial break at origin of circle
 other (255) } , -- something else
 alt SET OF INTEGER } -- set of alternatives for the integer
```

```
--**** User-object **********************************************
--*
--* a general object for a user defined structured data item
--* used by Seq-feat and Seq-descr

User-object ::= SEQUENCE {
 class VisibleString OPTIONAL , -- endeavor which designed this object
 type Object-id , -- type of object within class
 data SEQUENCE OF User-field } -- the object itself

User-field ::= SEQUENCE {
 label Object-id , -- field label
 num INTEGER OPTIONAL , -- required for strs, ints, reals, oss
 data CHOICE { -- field contents
 str VisibleString ,
 int INTEGER ,
 real REAL ,
 bool BOOLEAN ,
 os OCTET STRING ,
 object User-object , -- for using other definitions
 strs SEQUENCE OF VisibleString ,
 ints SEQUENCE OF INTEGER ,
 reals SEQUENCE OF REAL ,
 oss SEQUENCE OF OCTET STRING ,
 fields SEQUENCE OF User-field ,
 objects SEQUENCE OF User-object } }



END
```

ASN.1 Specification: biblio.asn

See also the online-version of this specification, which may be more up-to-date.

```
--$Revision$
--****************************************************************
--
-- NCBI Bibliographic data elements
-- by James Ostell, 1990
--
-- Taken from the American National Standard for
-- Bibliographic References
-- ANSI Z39.29-1977
-- Version 3.0 - June 1994
-- PubMedId added in 1996
-- ArticleIds and eprint elements added in 1999
--
--****************************************************************

NCBI-Biblio DEFINITIONS ::=
BEGIN

EXPORTS Cit-art, Cit-jour, Cit-book, Cit-pat, Cit-let, Id-pat, Cit-gen,
 Cit-proc, Cit-sub, Title, Author, PubMedId;

IMPORTS Person-id, Date, Dbtag FROM NCBI-General;

 -- Article Ids

ArticleId ::= CHOICE { -- can be many ids for an article
 pubmed PubMedId , -- see types below
 medline MedlineUID ,
 doi DOI ,
 pii PII ,
 pmcid PmcID ,
 pmcpid PmcPid ,
 pmpid PmPid ,
 other Dbtag } -- generic catch all

PubMedId ::= INTEGER -- Id from the PubMed database at NCBI
MedlineUID ::= INTEGER -- Id from MEDLINE
DOI ::= VisibleString -- Document Object Identifier
PII ::= VisibleString -- Controlled Publisher Identifier
PmcID ::= INTEGER -- PubMed Central Id
PmcPid ::= VisibleString -- Publisher Id supplied to PubMed Central
PmPid ::= VisibleString -- Publisher Id supplied to PubMed

ArticleIdSet ::= SET OF ArticleId

 -- Status Dates

PubStatus ::= INTEGER { -- points of publication
 received (1) , -- date manuscript received for review
 accepted (2) , -- accepted for publication
 epublish (3) , -- published electronically by publisher
 ppublish (4) , -- published in print by publisher
 revised (5) , -- article revised by publisher/author
 pmc (6) , -- article first appeared in PubMed Central
 pmcr (7) , -- article revision in PubMed Central
 pubmed (8) , -- article citation first appeared in PubMed
 pubmedr (9) , -- article citation revision in PubMed
 aheadofprint (10), -- epublish, but will be followed by print
 premedline (11), -- date into PreMedline status
 medline (12), -- date made a MEDLINE record
 other (255) }

PubStatusDate ::= SEQUENCE { -- done as a structure so fields can be added
 pubstatus PubStatus ,
 date Date } -- time may be added later

PubStatusDateSet ::= SET OF PubStatusDate
```

```
 -- Citation Types

Cit-art ::= SEQUENCE { -- article in journal or book
 title Title OPTIONAL , -- title of paper (ANSI requires)
 authors Auth-list OPTIONAL , -- authors (ANSI requires)
 from CHOICE { -- journal or book
 journal Cit-jour ,
 book Cit-book ,
 proc Cit-proc } ,
 ids ArticleIdSet OPTIONAL } -- lots of ids

Cit-jour ::= SEQUENCE { -- Journal citation
 title Title , -- title of journal
 imp Imprint }

Cit-book ::= SEQUENCE { -- Book citation
 title Title , -- Title of book
 coll Title OPTIONAL , -- part of a collection
 authors Auth-list, -- authors
 imp Imprint }

Cit-proc ::= SEQUENCE { -- Meeting proceedings
 book Cit-book , -- citation to meeting
 meet Meeting } -- time and location of meeting

 -- Patent number and date-issue were made optional in 1997 to
 -- support patent applications being issued from the USPTO
 -- Semantically a Cit-pat must have either a patent number or
 -- an application number (or both) to be valid

Cit-pat ::= SEQUENCE { -- patent citation
 title VisibleString ,
 authors Auth-list, -- author/inventor
 country VisibleString , -- Patent Document Country
 doc-type VisibleString , -- Patent Document Type
 number VisibleString OPTIONAL, -- Patent Document Number
 date-issue Date OPTIONAL, -- Patent Issue/Pub Date
 class SEQUENCE OF VisibleString OPTIONAL , -- Patent Doc Class Code
 app-number VisibleString OPTIONAL , -- Patent Doc Appl Number
 app-date Date OPTIONAL , -- Patent Appl File Date
 applicants Auth-list OPTIONAL , -- Applicants
 assignees Auth-list OPTIONAL , -- Assignees
 priority SEQUENCE OF Patent-priority OPTIONAL , -- Priorities
 abstract VisibleString OPTIONAL } -- abstract of patent

Patent-priority ::= SEQUENCE {
 country VisibleString , -- Patent country code
 number VisibleString , -- number assigned in that country
 date Date } -- date of application
```

```
Id-pat ::= SEQUENCE { -- just to identify a patent
 country VisibleString , -- Patent Document Country
 id CHOICE {
 number VisibleString , -- Patent Document Number
 app-number VisibleString } , -- Patent Doc Appl Number
 doc-type VisibleString OPTIONAL } -- Patent Doc Type

Cit-let ::= SEQUENCE { -- letter, thesis, or manuscript
 cit Cit-book , -- same fields as a book
 man-id VisibleString OPTIONAL , -- Manuscript identifier
 type ENUMERATED {
 manuscript (1) ,
 letter (2) ,
 thesis (3) } OPTIONAL }
 -- NOTE: this is just to cite a
 -- direct data submission, see NCBI-Submit
 -- for the form of a sequence submission
Cit-sub ::= SEQUENCE { -- citation for a direct submission
 authors Auth-list , -- not necessarily authors of the paper
 imp Imprint OPTIONAL , -- this only used to get date.. will go
 medium ENUMERATED { -- medium of submission
 paper (1) ,
 tape (2) ,
 floppy (3) ,
 email (4) ,
 other (255) } OPTIONAL ,
 date Date OPTIONAL , -- replaces imp, will become required
 descr VisibleString OPTIONAL } -- description of changes for public view

Cit-gen ::= SEQUENCE { -- NOT from ANSI, this is a catchall
 cit VisibleString OPTIONAL , -- anything, not parsable
 authors Auth-list OPTIONAL ,
 muid INTEGER OPTIONAL , -- medline uid
 journal Title OPTIONAL ,
 volume VisibleString OPTIONAL ,
 issue VisibleString OPTIONAL ,
 pages VisibleString OPTIONAL ,
 date Date OPTIONAL ,
 serial-number INTEGER OPTIONAL , -- for GenBank style references
 title VisibleString OPTIONAL , -- eg. cit="unpublished",title="title"
 pmid PubMedId OPTIONAL } -- PubMed Id


 -- Authorship Group
Auth-list ::= SEQUENCE {
 names CHOICE {
 std SEQUENCE OF Author , -- full citations
 ml SEQUENCE OF VisibleString , -- MEDLINE, semi-structured
 str SEQUENCE OF VisibleString } , -- free for all
 affil Affil OPTIONAL } -- author affiliation
```

```
Author ::= SEQUENCE {
 name Person-id , -- Author, Primary or Secondary
 level ENUMERATED {
 primary (1),
 secondary (2) } OPTIONAL ,
 role ENUMERATED { -- Author Role Indicator
 compiler (1),
 editor (2),
 patent-assignee (3),
 translator (4) } OPTIONAL ,
 affil Affil OPTIONAL ,
 is-corr BOOLEAN OPTIONAL } -- TRUE if corresponding author

Affil ::= CHOICE {
 str VisibleString , -- unparsed string
 std SEQUENCE { -- std representation
 affil VisibleString OPTIONAL , -- Author Affiliation, Name
 div VisibleString OPTIONAL , -- Author Affiliation, Division
 city VisibleString OPTIONAL , -- Author Affiliation, City
 sub VisibleString OPTIONAL , -- Author Affiliation, County Sub
 country VisibleString OPTIONAL , -- Author Affiliation, Country
 street VisibleString OPTIONAL , -- street address, not ANSI
 email VisibleString OPTIONAL ,
 fax VisibleString OPTIONAL ,
 phone VisibleString OPTIONAL ,
 postal-code VisibleString OPTIONAL }}

 -- Title Group
 -- Valid for = A = Analytic (Cit-art)
 -- J = Journals (Cit-jour)
 -- B = Book (Cit-book)
 -- Valid for:
Title ::= SET OF CHOICE {
 name VisibleString , -- Title, Anal,Coll,Mono AJB
 tsub VisibleString , -- Title, Subordinate A B
 trans VisibleString , -- Title, Translated AJB
 jta VisibleString , -- Title, Abbreviated J
 iso-jta VisibleString , -- specifically ISO jta J
 ml-jta VisibleString , -- specifically MEDLINE jta J
 coden VisibleString , -- a coden J
 issn VisibleString , -- ISSN J
 abr VisibleString , -- Title, Abbreviated B
 isbn VisibleString } -- ISBN B

Imprint ::= SEQUENCE { -- Imprint group
 date Date , -- date of publication
 volume VisibleString OPTIONAL ,
 issue VisibleString OPTIONAL ,
 pages VisibleString OPTIONAL ,
 section VisibleString OPTIONAL ,
 pub Affil OPTIONAL, -- publisher, required for book
 cprt Date OPTIONAL, -- copyright date, " " "
 part-sup VisibleString OPTIONAL , -- part/sup of volume
 language VisibleString DEFAULT "ENG" , -- put here for simplicity
 prepub ENUMERATED { -- for prepublication citations
 submitted (1) , -- submitted, not accepted
 in-press (2) , -- accepted, not published
 other (255) } OPTIONAL ,
 part-supi VisibleString OPTIONAL , -- part/sup on issue
 retract CitRetract OPTIONAL , -- retraction info
 pubstatus PubStatus OPTIONAL , -- current status of this publication
 history PubStatusDateSet OPTIONAL } -- dates for this record

CitRetract ::= SEQUENCE {
 type ENUMERATED { -- retraction of an entry
 retracted (1) , -- this citation retracted
 notice (2) , -- this citation is a retraction notice
 in-error (3) , -- an erratum was published about this
 erratum (4) } , -- this is a published erratum
 exp VisibleString OPTIONAL } -- citation and/or explanation

Meeting ::= SEQUENCE {
 number VisibleString ,
 date Date ,
 place Affil OPTIONAL }

END
```

ASN.1 Specification: pub.asn

See also the online-version of this specification, which may be more up-to-date.

```
--$Revision$
--**********************************************************************
--
-- Publication common set
-- James Ostell, 1990
--
-- This is the base class definitions for Publications of all sorts
--
-- support for PubMedId added in 1996
--**********************************************************************

NCBI-Pub DEFINITIONS ::=
BEGIN

EXPORTS Pub, Pub-set, Pub-equiv;

IMPORTS Medline-entry FROM NCBI-Medline
 Cit-art, Cit-jour, Cit-book, Cit-proc, Cit-pat, Id-pat, Cit-gen,
 Cit-let, Cit-sub, PubMedId FROM NCBI-Biblio;

Pub ::= CHOICE {
 gen Cit-gen , -- general or generic unparsed
 sub Cit-sub , -- submission
 medline Medline-entry ,
 muid INTEGER , -- medline uid
 article Cit-art ,
 journal Cit-jour ,
 book Cit-book ,
 proc Cit-proc , -- proceedings of a meeting
 patent Cit-pat ,
 pat-id Id-pat , -- identify a patent
 man Cit-let , -- manuscript, thesis, or letter
 equiv Pub-equiv, -- to cite a variety of ways
 pmid PubMedId } -- PubMedId

Pub-equiv ::= SET OF Pub -- equivalent identifiers for same citation

Pub-set ::= CHOICE {
 pub SET OF Pub ,
 medline SET OF Medline-entry ,
 article SET OF Cit-art ,
 journal SET OF Cit-jour ,
 book SET OF Cit-book ,
 proc SET OF Cit-proc , -- proceedings of a meeting
 patent SET OF Cit-pat }

END
```

ASN.1 Specification: medline.asn

See also the online-version of this specification, which may be more up-to-date.

```
--$Revision$
--**********************************************************************
--
-- MEDLINE data definitions
-- James Ostell, 1990
--
-- enhanced in 1996 to support PubMed records as well by simply adding
-- the PubMedId and making MedlineId optional
--
--**********************************************************************

NCBI-Medline DEFINITIONS ::=
BEGIN

EXPORTS Medline-entry, Medline-si;

IMPORTS Cit-art, PubMedId FROM NCBI-Biblio
 Date FROM NCBI-General;

 -- a MEDLINE or PubMed entry
Medline-entry ::= SEQUENCE {
 uid INTEGER OPTIONAL , -- MEDLINE UID, sometimes not yet available if from
PubMed
 em Date , -- Entry Month
 cit Cit-art , -- article citation
 abstract VisibleString OPTIONAL ,
 mesh SET OF Medline-mesh OPTIONAL ,
 substance SET OF Medline-rn OPTIONAL ,
 xref SET OF Medline-si OPTIONAL ,
 idnum SET OF VisibleString OPTIONAL , -- ID Number (grants, contracts)
 gene SET OF VisibleString OPTIONAL ,
 pmid PubMedId OPTIONAL , -- MEDLINE records may include the PubMedId
 pub-type SET OF VisibleString OPTIONAL, -- may show publication types
(review, etc)
 mlfield SET OF Medline-field OPTIONAL , -- additional Medline field types
 status INTEGER {
 publisher (1) , -- record as supplied by publisher
 premedline (2) , -- premedline record
 medline (3) } DEFAULT medline } -- regular medline record

Medline-mesh ::= SEQUENCE {
 mp BOOLEAN DEFAULT FALSE , -- TRUE if main point (*)
 term VisibleString , -- the MeSH term
 qual SET OF Medline-qual OPTIONAL } -- qualifiers

Medline-qual ::= SEQUENCE {
 mp BOOLEAN DEFAULT FALSE , -- TRUE if main point
 subh VisibleString } -- the subheading

Medline-rn ::= SEQUENCE { -- medline substance records
 type ENUMERATED { -- type of record
 nameonly (0) ,
 cas (1) , -- CAS number
 ec (2) } , -- EC number
 cit VisibleString OPTIONAL , -- CAS or EC number if present
 name VisibleString } -- name (always present)

Medline-si ::= SEQUENCE { -- medline cross reference records
 type ENUMERATED { -- type of xref
 ddbj (1) , -- DNA Data Bank of Japan
 carbbank (2) , -- Carbohydrate Structure Database
 embl (3) , -- EMBL Data Library
 hdb (4) , -- Hybridoma Data Bank
 genbank (5) , -- GenBank
 hgml (6) , -- Human Gene Map Library
 mim (7) , -- Mendelian Inheritance in Man
 msd (8) , -- Microbial Strains Database
 pdb (9) , -- Protein Data Bank (Brookhaven)
 pir (10) , -- Protein Identification Resource
 prfseqdb (11) , -- Protein Research Foundation (Japan)
 psd (12) , -- Protein Sequence Database (Japan)
 swissprot (13) , -- SwissProt
 gdb (14) } , -- Genome Data Base
 cit VisibleString OPTIONAL } -- the citation/accession number

Medline-field ::= SEQUENCE {
 type INTEGER { -- Keyed type
 other (0) , -- look in line code
 comment (1) , -- comment line
 erratum (2) } , -- retracted, corrected, etc
 str VisibleString , -- the text
 ids SEQUENCE OF DocRef OPTIONAL } -- pointers relevant to this text

DocRef ::= SEQUENCE { -- reference to a document
 type INTEGER {
 medline (1) ,
 pubmed (2) ,
 ncbigi (3) } ,
 uid INTEGER }

END
```

ASN.1 Specification: seq.asn

See also the online-version of this specification, which may be more up-to-date.

```
--$Revision$
--**********************************************************************
--
-- NCBI Sequence elements
-- by James Ostell, 1990
-- Version 3.0 - June 1994
--
--**********************************************************************

NCBI-Sequence DEFINITIONS ::=
BEGIN

EXPORTS Annotdesc, Annot-descr, Bioseq, GIBB-mol, Heterogen, MolInfo,
 Numbering, Pubdesc, Seq-annot, Seq-data, Seqdesc, Seq-descr, Seq-ext,
 Seq-hist, Seq-inst, Seq-literal, Seqdesc, Delta-ext, Seq-gap;

IMPORTS Date, Int-fuzz, Dbtag, Object-id, User-object FROM NCBI-General
 Seq-align FROM NCBI-Seqalign
 Seq-feat FROM NCBI-Seqfeat
 Seq-graph FROM NCBI-Seqres
 Pub-equiv FROM NCBI-Pub
 Org-ref FROM NCBI-Organism
 BioSource FROM NCBI-BioSource
 Seq-id, Seq-loc FROM NCBI-Seqloc
 GB-block FROM GenBank-General
 PIR-block FROM PIR-General
 EMBL-block FROM EMBL-General
 SP-block FROM SP-General
 PRF-block FROM PRF-General
 PDB-block FROM PDB-General
 Seq-table FROM NCBI-SeqTable;

--*** Sequence *********************************
--*

Bioseq ::= SEQUENCE {
 id SET OF Seq-id , -- equivalent identifiers
 descr Seq-descr OPTIONAL , -- descriptors
 inst Seq-inst , -- the sequence data
 annot SET OF Seq-annot OPTIONAL }

--*** Descriptors *****************************
--*

Seq-descr ::= SET OF Seqdesc

Seqdesc ::= CHOICE {
 mol-type GIBB-mol , -- type of molecule
 modif SET OF GIBB-mod , -- modifiers
 method GIBB-method , -- sequencing method
 name VisibleString , -- a name for this sequence
 title VisibleString , -- a title for this sequence
 org Org-ref , -- if all from one organism
 comment VisibleString , -- a more extensive comment
 num Numbering , -- a numbering system
 maploc Dbtag , -- map location of this sequence
 pir PIR-block , -- PIR specific info
 genbank GB-block , -- GenBank specific info
 pub Pubdesc , -- a reference to the publication
 region VisibleString , -- overall region (globin locus)
 user User-object , -- user defined object
 sp SP-block , -- SWISSPROT specific info
 dbxref Dbtag , -- xref to other databases
 embl EMBL-block , -- EMBL specific information
 create-date Date , -- date entry first created/released
 update-date Date , -- date of last update
 prf PRF-block , -- PRF specific information
 pdb PDB-block , -- PDB specific information
 het Heterogen , -- cofactor, etc associated but not bound
 source BioSource , -- source of materials, includes Org-ref
 molinfo MolInfo } -- info on the molecule and techniques

--******* NOTE:
--* mol-type, modif, method, and org are consolidated and expanded
--* in Org-ref, BioSource, and MolInfo in this specification. They
--* will be removed in later specifications. Do not use them in the
--* the future. Instead expect the new structures.
--*
--***************************
```

```
--**********************************************************************
--
-- MolInfo gives information on the
-- classification of the type and quality of the sequence
--
-- WARNING: this will replace GIBB-mol, GIBB-mod, GIBB-method
--
--**********************************************************************

MolInfo ::= SEQUENCE {
 biomol INTEGER {
 unknown (0) ,
 genomic (1) ,
 pre-RNA (2) , -- precursor RNA of any sort really
 mRNA (3) ,
 rRNA (4) ,
 tRNA (5) ,
 snRNA (6) ,
 scRNA (7) ,
 peptide (8) ,
 other-genetic (9) , -- other genetic material
 genomic-mRNA (10) , -- reported a mix of genomic and cdna sequence
 cRNA (11) , -- viral RNA genome copy intermediate
 snoRNA (12) , -- small nucleolar RNA
 transcribed-RNA (13) , -- transcribed RNA other than existing classes
 ncRNA (14) ,
 tmRNA (15) ,
 other (255) } DEFAULT unknown ,
 tech INTEGER {
 unknown (0) ,
 standard (1) , -- standard sequencing
 est (2) , -- Expressed Sequence Tag
 sts (3) , -- Sequence Tagged Site
 survey (4) , -- one-pass genomic sequence
 genemap (5) , -- from genetic mapping techniques
 physmap (6) , -- from physical mapping techniques
 derived (7) , -- derived from other data, not a primary entity
 concept-trans (8) , -- conceptual translation
 seq-pept (9) , -- peptide was sequenced
 both (10) , -- concept transl. w/ partial pept. seq.
 seq-pept-overlap (11) , -- sequenced peptide, ordered by overlap
 seq-pept-homol (12) , -- sequenced peptide, ordered by homology
 concept-trans-a (13) , -- conceptual transl. supplied by author
 htgs-1 (14) , -- unordered High Throughput sequence contig
 htgs-2 (15) , -- ordered High Throughput sequence contig
 htgs-3 (16) , -- finished High Throughput sequence
 fli-cdna (17) , -- full length insert cDNA
 htgs-0 (18) , -- single genomic reads for coordination
 htc (19) , -- high throughput cDNA
 wgs (20) , -- whole genome shotgun sequencing
 barcode (21) , -- barcode of life project
 composite-wgs-htgs (22) , -- composite of WGS and HTGS
 tsa (23) , -- transcriptome shotgun assembly
 other (255) } -- use Source.techexp
 DEFAULT unknown ,
 techexp VisibleString OPTIONAL , -- explanation if tech not enough
 --
 -- Completeness is not indicated in most records. For genomes, assume
 -- the sequences are incomplete unless specifically marked as complete.
 -- For mRNAs, assume the ends are not known exactly unless marked as
 -- having the left or right end.
 --
 completeness INTEGER {
 unknown (0) ,
 complete (1) , -- complete biological entity
 partial (2) , -- partial but no details given
 no-left (3) , -- missing 5' or NH3 end
 no-right (4) , -- missing 3' or COOH end
 no-ends (5) , -- missing both ends
 has-left (6) , -- 5' or NH3 end present
 has-right (7) , -- 3' or COOH end present
 other (255) } DEFAULT unknown ,
 gbmoltype VisibleString OPTIONAL } -- identifies particular ncRNA


GIBB-mol ::= ENUMERATED { -- type of molecule represented
 unknown (0) ,
 genomic (1) ,
 pre-mRNA (2) , -- precursor RNA of any sort really
 mRNA (3) ,
 rRNA (4) ,
 tRNA (5) ,
 snRNA (6) ,
 scRNA (7) ,
 peptide (8) ,
 other-genetic (9) , -- other genetic material
 genomic-mRNA (10) , -- reported a mix of genomic and cdna sequence
 other (255) }

GIBB-mod ::= ENUMERATED { -- GenInfo Backbone modifiers
 dna (0) ,
 rna (1) ,
 extrachrom (2) ,
 plasmid (3) ,
 mitochondrial (4) ,
 chloroplast (5) ,
 kinetoplast (6) ,
 cyanelle (7) ,
 synthetic (8) ,
 recombinant (9) ,
 partial (10) ,
 complete (11) ,
 mutagen (12) , -- subject of mutagenesis ?
 natmut (13) , -- natural mutant ?
 transposon (14) ,
 insertion-seq (15) ,
 no-left (16) , -- missing left end (5' for na, NH2 for aa)
 no-right (17) , -- missing right end (3' or COOH)
 macronuclear (18) ,
 proviral (19) ,
 est (20) , -- expressed sequence tag
 sts (21) , -- sequence tagged site
 survey (22) , -- one pass survey sequence
 chromoplast (23) ,
 genemap (24) , -- is a genetic map
 restmap (25) , -- is an ordered restriction map
 physmap (26) , -- is a physical map (not ordered restriction map)
 other (255) }

GIBB-method ::= ENUMERATED { -- sequencing methods
 concept-trans (1) , -- conceptual translation
 seq-pept (2) , -- peptide was sequenced
 both (3) , -- concept transl. w/ partial pept. seq.
 seq-pept-overlap (4) , -- sequenced peptide, ordered by overlap
 seq-pept-homol (5) , -- sequenced peptide, ordered by homology
 concept-trans-a (6) , -- conceptual transl. supplied by author
 other (255) }

Numbering ::= CHOICE { -- any display numbering system
 cont Num-cont , -- continuous numbering
 enum Num-enum , -- enumerated names for residues
 ref Num-ref , -- by reference to another sequence
 real Num-real } -- supports mapping to a float system

Num-cont ::= SEQUENCE { -- continuous display numbering system
 refnum INTEGER DEFAULT 1, -- number assigned to first residue
 has-zero BOOLEAN DEFAULT FALSE , -- 0 used?
 ascending BOOLEAN DEFAULT TRUE } -- ascending numbers?

Num-enum ::= SEQUENCE { -- any tags to residues
 num INTEGER , -- number of tags to follow
 names SEQUENCE OF VisibleString } -- the tags

Num-ref ::= SEQUENCE { -- by reference to other sequences
 type ENUMERATED { -- type of reference
 not-set (0) ,
 sources (1) , -- by segmented or const seq sources
 aligns (2) } , -- by alignments given below
 aligns Seq-align OPTIONAL }

Num-real ::= SEQUENCE { -- mapping to floating point system
 a REAL , -- from an integer system used by Bioseq
 b REAL , -- position = (a * int_position) + b
 units VisibleString OPTIONAL }

Pubdesc ::= SEQUENCE { -- how sequence presented in pub
 pub Pub-equiv , -- the citation(s)
 name VisibleString OPTIONAL , -- name used in paper
 fig VisibleString OPTIONAL , -- figure in paper
 num Numbering OPTIONAL , -- numbering from paper
 numexc BOOLEAN OPTIONAL , -- numbering problem with paper
 poly-a BOOLEAN OPTIONAL , -- poly A tail indicated in figure?
 maploc VisibleString OPTIONAL , -- map location reported in paper
 seq-raw StringStore OPTIONAL , -- original sequence from paper
 align-group INTEGER OPTIONAL , -- this seq aligned with others in paper
 comment VisibleString OPTIONAL, -- any comment on this pub in context
 reftype INTEGER { -- type of reference in a GenBank record
 seq (0) , -- refers to sequence
 sites (1) , -- refers to unspecified features
 feats (2) , -- refers to specified features
 no-target (3) } -- nothing specified (EMBL)
 DEFAULT seq }

Heterogen ::= VisibleString -- cofactor, prosthetic group, inhibitor, etc

--*** Instances of sequences *******************************
--*

Seq-inst ::= SEQUENCE { -- the sequence data itself
 repr ENUMERATED { -- representation class
 not-set (0) , -- empty
 virtual (1) , -- no seq data
 raw (2) , -- continuous sequence
 seg (3) , -- segmented sequence
 const (4) , -- constructed sequence
 ref (5) , -- reference to another sequence
 consen (6) , -- consensus sequence or pattern
 map (7) , -- ordered map of any kind
 delta (8) , -- sequence made by changes (delta) to others
 other (255) } ,
 mol ENUMERATED { -- molecule class in living organism
 not-set (0) , -- > cdna = rna
 dna (1) ,
 rna (2) ,
 aa (3) ,
 na (4) , -- just a nucleic acid
 other (255) } ,
 length INTEGER OPTIONAL , -- length of sequence in residues
 fuzz Int-fuzz OPTIONAL , -- length uncertainty
 topology ENUMERATED { -- topology of molecule
 not-set (0) ,
 linear (1) ,
 circular (2) ,
 tandem (3) , -- some part of tandem repeat
 other (255) } DEFAULT linear ,
 strand ENUMERATED { -- strandedness in living organism
 not-set (0) ,
 ss (1) , -- single strand
 ds (2) , -- double strand
 mixed (3) ,
 other (255) } OPTIONAL , -- default ds for DNA, ss for RNA, pept
 seq-data Seq-data OPTIONAL , -- the sequence
 ext Seq-ext OPTIONAL , -- extensions for special types
 hist Seq-hist OPTIONAL } -- sequence history

--*** Sequence Extensions **********************************
--* for representing more complex types
--* const type uses Seq-hist.assembly

Seq-ext ::= CHOICE {
 seg Seg-ext , -- segmented sequences
 ref Ref-ext , -- hot link to another sequence (a view)
 map Map-ext , -- ordered map of markers
 delta Delta-ext }

Seg-ext ::= SEQUENCE OF Seq-loc

Ref-ext ::= Seq-loc

Map-ext ::= SEQUENCE OF Seq-feat

Delta-ext ::= SEQUENCE OF Delta-seq

Delta-seq ::= CHOICE {
 loc Seq-loc , -- point to a sequence
 literal Seq-literal } -- a piece of sequence

Seq-literal ::= SEQUENCE {
 length INTEGER , -- must give a length in residues
 fuzz Int-fuzz OPTIONAL , -- could be unsure
 seq-data Seq-data OPTIONAL } -- may have the data

--*** Sequence History Record ***********************************
--** assembly = records how seq was assembled from others
--** replaces = records sequences made obsolete by this one
--** replaced-by = this seq is made obsolete by another(s)

Seq-hist ::= SEQUENCE {
 assembly SET OF Seq-align OPTIONAL ,-- how was this assembled?
 replaces Seq-hist-rec OPTIONAL , -- seq makes these seqs obsolete
 replaced-by Seq-hist-rec OPTIONAL , -- these seqs make this one obsolete
 deleted CHOICE {
 bool BOOLEAN ,
 date Date } OPTIONAL }
```

```
Seq-hist-rec ::= SEQUENCE {
 date Date OPTIONAL ,
 ids SET OF Seq-id }

--*** Various internal sequence representations ************
--* all are controlled, fixed length forms

Seq-data ::= CHOICE { -- sequence representations
 iupacna IUPACna , -- IUPAC 1 letter nuc acid code
 iupacaa IUPACaa , -- IUPAC 1 letter amino acid code
 ncbi2na NCBI2na , -- 2 bit nucleic acid code
 ncbi4na NCBI4na , -- 4 bit nucleic acid code
 ncbi8na NCBI8na , -- 8 bit extended nucleic acid code
 ncbipna NCBIpna , -- nucleic acid probabilities
 ncbi8aa NCBI8aa , -- 8 bit extended amino acid codes
 ncbieaa NCBIeaa , -- extended ASCII 1 letter aa codes
 ncbipaa NCBIpaa , -- amino acid probabilities
 ncbistdaa NCBIstdaa, -- consecutive codes for std aas
 gap Seq-gap -- gap types
}

Seq-gap ::= SEQUENCE {
 type INTEGER {
 unknown(0),
 fragment(1),
 clone(2),
 short-arm(3),
 heterochromatin(4),
 centromere(5),
 telomere(6),
 repeat(7),
 contig(8),
 other(255)
 },
 linkage INTEGER {
 unlinked(0),
 linked(1),
 other(255)
 } OPTIONAL
}

IUPACna ::= StringStore -- IUPAC 1 letter codes, no spaces
IUPACaa ::= StringStore -- IUPAC 1 letter codes, no spaces
NCBI2na ::= OCTET STRING -- 00=A, 01=C, 10=G, 11=T
NCBI4na ::= OCTET STRING -- 1 bit each for agct
 -- 0001=A, 0010=C, 0100=G, 1000=T/U
 -- 0101=Purine, 1010=Pyrimidine, etc
NCBI8na ::= OCTET STRING -- for modified nucleic acids
NCBIpna ::= OCTET STRING -- 5 octets/base, prob for a,c,g,t,n
 -- probabilities are coded 0-255 = 0.0-1.0
```

```
NCBI8aa ::= OCTET STRING -- for modified amino acids
NCBIeaa ::= StringStore -- ASCII extended 1 letter aa codes
 -- IUPAC codes + U=selenocysteine
NCBIpaa ::= OCTET STRING -- 25 octets/aa, prob for IUPAC aas in order:
 -- A-Y,B,Z,X,(ter),anything
 -- probabilities are coded 0-255 = 0.0-1.0
NCBIstdaa ::= OCTET STRING -- codes 0-25, 1 per byte

--*** Sequence Annotation *************************************
--*

-- This is a replica of Textseq-id
-- This is specific for annotations, and exists to maintain a semantic
-- difference between IDs assigned to annotations and IDs assigned to
-- sequences
Textannot-id ::= SEQUENCE {
 name VisibleString OPTIONAL ,
 accession VisibleString OPTIONAL ,
 release VisibleString OPTIONAL ,
 version INTEGER OPTIONAL
}

Annot-id ::= CHOICE {
 local Object-id ,
 ncbi INTEGER ,
 general Dbtag,
 other Textannot-id
}

Annot-descr ::= SET OF Annotdesc

Annotdesc ::= CHOICE {
 name VisibleString , -- a short name for this collection
 title VisibleString , -- a title for this collection
 comment VisibleString , -- a more extensive comment
 pub Pubdesc , -- a reference to the publication
 user User-object , -- user defined object
 create-date Date , -- date entry first created/released
 update-date Date , -- date of last update
 src Seq-id , -- source sequence from which annot came
 align Align-def, -- definition of the SeqAligns
 region Seq-loc } -- all contents cover this region

Align-def ::= SEQUENCE {
 align-type INTEGER { -- class of align Seq-annot
 ref (1) , -- set of alignments to the same sequence
 alt (2) , -- set of alternate alignments of the same seqs
 blocks (3) , -- set of aligned blocks in the same seqs
 other (255) } ,
 ids SET OF Seq-id OPTIONAL } -- used for the one ref seqid for now
```

```
Seq-annot ::= SEQUENCE {
 id SET OF Annot-id OPTIONAL ,
 db INTEGER { -- source of annotation
 genbank (1) ,
 embl (2) ,
 ddbj (3) ,
 pir (4) ,
 sp (5) ,
 bbone (6) ,
 pdb (7) ,
 other (255) } OPTIONAL ,
 name VisibleString OPTIONAL ,-- source if "other" above
 desc Annot-descr OPTIONAL , -- used only for stand alone Seq-annots
 data CHOICE {
 ftable SET OF Seq-feat ,
 align SET OF Seq-align ,
 graph SET OF Seq-graph ,
 ids SET OF Seq-id , -- used for communication between tools
 locs SET OF Seq-loc , -- used for communication between tools
 seq-table Seq-table } } -- features in table form

END
```

ASN.1 Specification: seqblock.asn

See also the online-version of this specification, which may be more up-to-date.

```
--$Revision$
--**********************************************************************
--
-- 1990 - J.Ostell
-- Version 3.0 - June 1994
--
--**********************************************************************
--**********************************************************************
--
-- EMBL specific data
-- This block of specifications was developed by Reiner Fuchs of EMBL
-- Updated by J.Ostell, 1994
--
--**********************************************************************

EMBL-General DEFINITIONS ::=
BEGIN

EXPORTS EMBL-dbname, EMBL-xref, EMBL-block;

IMPORTS Date, Object-id FROM NCBI-General;

EMBL-dbname ::= CHOICE {
 code ENUMERATED {
 embl(0),
 genbank(1),
 ddbj(2),
 geninfo(3),
 medline(4),
 swissprot(5),
 pir(6),
 pdb(7),
 epd(8),
 ecd(9),
 tfd(10),
 flybase(11),
 prosite(12),
 enzyme(13),
 mim(14),
 ecoseq(15),
 hiv(16) ,
 other (255) } ,
 name VisibleString }

EMBL-xref ::= SEQUENCE {
 dbname EMBL-dbname,
 id SEQUENCE OF Object-id }

EMBL-block ::= SEQUENCE {
 class ENUMERATED {
 not-set(0),
 standard(1),
 unannotated(2),
 other(255) } DEFAULT standard,
 div ENUMERATED {
 fun(0),
 inv(1),
 mam(2),
 org(3),
 phg(4),
 pln(5),
 pri(6),
 pro(7),
 rod(8),
 syn(9),
 una(10),
 vrl(11),
 vrt(12),
 pat(13),
 est(14),
 sts(15),
 other (255) } OPTIONAL,
 creation-date Date,
 update-date Date,
 extra-acc SEQUENCE OF VisibleString OPTIONAL,
 keywords SEQUENCE OF VisibleString OPTIONAL,
 xref SEQUENCE OF EMBL-xref OPTIONAL }

END

--**********************************************************************
--
-- SWISSPROT specific data
-- This block of specifications was developed by Mark Cavanaugh of
-- NCBI working with Amos Bairoch of SWISSPROT
--
--**********************************************************************

SP-General DEFINITIONS ::=
BEGIN

EXPORTS SP-block;

IMPORTS Date, Dbtag FROM NCBI-General
 Seq-id FROM NCBI-Seqloc;

SP-block ::= SEQUENCE { -- SWISSPROT specific descriptions
 class ENUMERATED {
 not-set (0) ,
 standard (1) , -- conforms to all SWISSPROT checks
 prelim (2) , -- only seq and biblio checked
 other (255) } ,
 extra-acc SET OF VisibleString OPTIONAL , -- old SWISSPROT ids
 imeth BOOLEAN DEFAULT FALSE , -- seq known to start with Met
 plasnm SET OF VisibleString OPTIONAL, -- plasmid names carrying gene
 seqref SET OF Seq-id OPTIONAL, -- xref to other sequences
 dbref SET OF Dbtag OPTIONAL , -- xref to non-sequence dbases
 keywords SET OF VisibleString OPTIONAL , -- keywords
 created Date OPTIONAL , -- creation date
 sequpd Date OPTIONAL , -- sequence update
 annotupd Date OPTIONAL } -- annotation update

END

--**********************************************************************
--
-- PIR specific data
-- This block of specifications was developed by Jim Ostell of
-- NCBI
--
--**********************************************************************

PIR-General DEFINITIONS ::=
BEGIN

EXPORTS PIR-block;
```

```
IMPORTS Seq-id FROM NCBI-Seqloc;

PIR-block ::= SEQUENCE { -- PIR specific descriptions
 had-punct BOOLEAN OPTIONAL , -- had punctuation in sequence ?
 host VisibleString OPTIONAL ,
 source VisibleString OPTIONAL , -- source line
 summary VisibleString OPTIONAL ,
 genetic VisibleString OPTIONAL ,
 includes VisibleString OPTIONAL ,
 placement VisibleString OPTIONAL ,
 superfamily VisibleString OPTIONAL ,
 keywords SEQUENCE OF VisibleString OPTIONAL ,
 cross-reference VisibleString OPTIONAL ,
 date VisibleString OPTIONAL ,
 seq-raw VisibleString OPTIONAL , -- seq with punctuation
 seqref SET OF Seq-id OPTIONAL } -- xref to other sequences

END

--**********************************************************************
--
-- GenBank specific data
-- This block of specifications was developed by Jim Ostell of
-- NCBI
--
--**********************************************************************

GenBank-General DEFINITIONS ::=
BEGIN

EXPORTS GB-block;

IMPORTS Date FROM NCBI-General;

GB-block ::= SEQUENCE { -- GenBank specific descriptions
 extra-accessions SEQUENCE OF VisibleString OPTIONAL ,
 source VisibleString OPTIONAL , -- source line
 keywords SEQUENCE OF VisibleString OPTIONAL ,
 origin VisibleString OPTIONAL,
 date VisibleString OPTIONAL , -- OBSOLETE old form Entry Date
 entry-date Date OPTIONAL , -- replaces date
 div VisibleString OPTIONAL , -- GenBank division
 taxonomy VisibleString OPTIONAL } -- continuation line of organism

END

--***********************************************************************
-- PRF specific definition
-- PRF is a protein sequence database crated and maintained by
-- Protein Research Foundation, Minoo-city, Osaka, Japan.
--
```

```
-- Written by A.Ogiwara, Inst.Chem.Res. (Dr.Kanehisa's Lab),
-- Kyoto Univ., Japan
--
--**********************************************************************

PRF-General DEFINITIONS ::=
BEGIN

EXPORTS PRF-block;

PRF-block ::= SEQUENCE {
 extra-src PRF-ExtraSrc OPTIONAL,
 keywords SEQUENCE OF VisibleString OPTIONAL
}

PRF-ExtraSrc ::= SEQUENCE {
 host VisibleString OPTIONAL,
 part VisibleString OPTIONAL,
 state VisibleString OPTIONAL,
 strain VisibleString OPTIONAL,
 taxon VisibleString OPTIONAL
}

END

--*********************************************************************
--
-- PDB specific data
-- This block of specifications was developed by Jim Ostell and
-- Steve Bryant of NCBI
--
--*********************************************************************

PDB-General DEFINITIONS ::=
BEGIN

EXPORTS PDB-block;

IMPORTS Date FROM NCBI-General;

PDB-block ::= SEQUENCE { -- PDB specific descriptions
 deposition Date , -- deposition date month,year
 class VisibleString ,
 compound SEQUENCE OF VisibleString ,
 source SEQUENCE OF VisibleString ,
 exp-method VisibleString OPTIONAL , -- present if NOT X-ray diffraction
 replace PDB-replace OPTIONAL } -- replacement history

PDB-replace ::= SEQUENCE {
 date Date ,
 ids SEQUENCE OF VisibleString } -- entry ids replace by this one
```

```
END
```

ASN.1 Specification: seqcode.asn

See also the online-version of this specification, which may be more up-to-date.

```
--$Revision$
-- **********************************************************************
--
-- These are code and conversion tables for NCBI sequence codes
-- ASN.1 for the sequences themselves are define in seq.asn
--
-- Seq-map-table and Seq-code-table REQUIRE that codes start with 0
-- and increase continuously. So IUPAC codes, which are upper case
-- letters will always have 65 0 cells before the codes begin. This
-- allows all codes to do indexed lookups for things
--
-- Valid names for code tables are:
-- IUPACna
-- IUPACaa
-- IUPACeaa
-- IUPACaa3 3 letter amino acid codes : parallels IUPACeaa
-- display only, not a data exchange type
-- NCBI2na
-- NCBI4na
-- NCBI8na
-- NCBI8aa
-- NCBIstdaa
-- probability types map to IUPAC types for display as characters

NCBI-SeqCode DEFINITIONS ::=
BEGIN

EXPORTS Seq-code-table, Seq-map-table, Seq-code-set;

Seq-code-type ::= ENUMERATED { -- sequence representations
 iupacna (1) , -- IUPAC 1 letter nuc acid code
 iupacaa (2) , -- IUPAC 1 letter amino acid code
 ncbi2na (3) , -- 2 bit nucleic acid code
 ncbi4na (4) , -- 4 bit nucleic acid code
 ncbi8na (5) , -- 8 bit extended nucleic acid code
 ncbipna (6) , -- nucleic acid probabilities
 ncbi8aa (7) , -- 8 bit extended amino acid codes
 ncbieaa (8) , -- extended ASCII 1 letter aa codes
 ncbipaa (9) , -- amino acid probabilities
 iupacaa3 (10) , -- 3 letter code only for display
 ncbistdaa (11) } -- consecutive codes for std aas, 0-25

Seq-map-table ::= SEQUENCE { -- for tables of sequence mappings
 from Seq-code-type , -- code to map from
 to Seq-code-type , -- code to map to
 num INTEGER , -- number of rows in table
 start-at INTEGER DEFAULT 0 , -- index offset of first element
 table SEQUENCE OF INTEGER } -- table of values, in from-to order

Seq-code-table ::= SEQUENCE { -- for names of coded values
 code Seq-code-type , -- name of code
 num INTEGER , -- number of rows in table
 one-letter BOOLEAN , -- symbol is ALWAYS 1 letter?
 start-at INTEGER DEFAULT 0 , -- index offset of first element
 table SEQUENCE OF
 SEQUENCE {
 symbol VisibleString , -- the printed symbol or letter
 name VisibleString } , -- an explanatory name or string
 comps SEQUENCE OF INTEGER OPTIONAL } -- pointers to complement nuc acid

Seq-code-set ::= SEQUENCE { -- for distribution
 codes SET OF Seq-code-table OPTIONAL ,
 maps SET OF Seq-map-table OPTIONAL }

END
```

ASN.1 Specification: seqset.asn

See also the online-version of this specification, which may be more up-to-date.

```
--$Revision$
--**********************************************************************
--
-- NCBI Sequence Collections
-- by James Ostell, 1990
--
-- Version 3.0 - 1994
--
--**********************************************************************

NCBI-Seqset DEFINITIONS ::=
BEGIN

EXPORTS Bioseq-set, Seq-entry;

IMPORTS Bioseq, Seq-annot, Seq-descr FROM NCBI-Sequence
 Object-id, Dbtag, Date FROM NCBI-General;

--*** Sequence Collections ********************************
--*

Bioseq-set ::= SEQUENCE { -- just a collection
 id Object-id OPTIONAL ,
 coll Dbtag OPTIONAL , -- to identify a collection
 level INTEGER OPTIONAL , -- nesting level
 class ENUMERATED {
 not-set (0) ,
 nuc-prot (1) , -- nuc acid and coded proteins
 segset (2) , -- segmented sequence + parts
 conset (3) , -- constructed sequence + parts
 parts (4) , -- parts for 2 or 3
 gibb (5) , -- geninfo backbone
 gi (6) , -- geninfo
 genbank (7) , -- converted genbank
 pir (8) , -- converted pir
 pub-set (9) , -- all the seqs from a single publication
 equiv (10) , -- a set of equivalent maps or seqs
 swissprot (11) , -- converted SWISSPROT
 pdb-entry (12) , -- a complete PDB entry
 mut-set (13) , -- set of mutations
 pop-set (14) , -- population study
 phy-set (15) , -- phylogenetic study
 eco-set (16) , -- ecological sample study
 gen-prod-set (17) , -- genomic products, chrom+mRNA+protein
 wgs-set (18) , -- whole genome shotgun project
 named-annot (19) , -- named annotation set
 named-annot-prod (20) , -- with instantiated mRNA+protein
 read-set (21) , -- set from a single read
 paired-end-reads (22) , -- paired sequences within a read-set
 other (255) } DEFAULT not-set ,
 release VisibleString OPTIONAL ,
 date Date OPTIONAL ,
 descr Seq-descr OPTIONAL ,
 seq-set SEQUENCE OF Seq-entry ,
 annot SET OF Seq-annot OPTIONAL }

Seq-entry ::= CHOICE {
 seq Bioseq ,
 set Bioseq-set }

END
```

ASN.1 Specification: seqloc.asn

See also the online-version of this specification, which may be more up-to-date.

```
--$Revision$
--**********************************************************************
--
-- NCBI Sequence location and identifier elements
-- by James Ostell, 1990
--
-- Version 3.0 - 1994
--
--**********************************************************************

NCBI-Seqloc DEFINITIONS ::=
BEGIN
```

```
EXPORTS Seq-id, Seq-loc, Seq-interval, Packed-seqint, Seq-point, Packed-
seqpnt,
 Na-strand, Giimport-id;

IMPORTS Object-id, Int-fuzz, Dbtag, Date FROM NCBI-General
 Id-pat FROM NCBI-Biblio
 Feat-id FROM NCBI-Seqfeat;

--*** Sequence identifiers ********************************
--*

Seq-id ::= CHOICE {
 local Object-id , -- local use
 gibbsq INTEGER , -- Geninfo backbone seqid
 gibbmt INTEGER , -- Geninfo backbone moltype
 giim Giimport-id , -- Geninfo import id
 genbank Textseq-id ,
 embl Textseq-id ,
 pir Textseq-id ,
 swissprot Textseq-id ,
 patent Patent-seq-id ,
 other Textseq-id , -- for historical reasons, 'other' = 'refseq'
 general Dbtag , -- for other databases
 gi INTEGER , -- GenInfo Integrated Database
 ddbj Textseq-id , -- DDBJ
 prf Textseq-id , -- PRF SEQDB
 pdb PDB-seq-id , -- PDB sequence
 tpg Textseq-id , -- Third Party Annot/Seq Genbank
 tpe Textseq-id , -- Third Party Annot/Seq EMBL
 tpd Textseq-id , -- Third Party Annot/Seq DDBJ
 gpipe Textseq-id , -- Internal NCBI genome pipeline processing ID
 named-annot-track Textseq-id -- Internal named annotation tracking ID
}

Seq-id-set ::= SET OF Seq-id


Patent-seq-id ::= SEQUENCE {
 seqid INTEGER , -- number of sequence in patent
 cit Id-pat } -- patent citation

Textseq-id ::= SEQUENCE {
 name VisibleString OPTIONAL ,
 accession VisibleString OPTIONAL ,
 release VisibleString OPTIONAL ,
 version INTEGER OPTIONAL }

Giimport-id ::= SEQUENCE {
 id INTEGER , -- the id to use here
 db VisibleString OPTIONAL , -- dbase used in
 release VisibleString OPTIONAL } -- the release
```

```
PDB-seq-id ::= SEQUENCE {
 mol PDB-mol-id , -- the molecule name
 chain INTEGER DEFAULT 32 , -- a single ASCII character, chain id
 rel Date OPTIONAL } -- release date, month and year

PDB-mol-id ::= VisibleString -- name of mol, 4 chars

--*** Sequence locations **********************************
--*

Seq-loc ::= CHOICE {
 null NULL , -- not placed
 empty Seq-id , -- to NULL one Seq-id in a collection
 whole Seq-id , -- whole sequence
 int Seq-interval , -- from to
 packed-int Packed-seqint ,
 pnt Seq-point ,
 packed-pnt Packed-seqpnt ,
 mix Seq-loc-mix ,
 equiv Seq-loc-equiv , -- equivalent sets of locations
 bond Seq-bond ,
 feat Feat-id } -- indirect, through a Seq-feat


Seq-interval ::= SEQUENCE {
 from INTEGER ,
 to INTEGER ,
 strand Na-strand OPTIONAL ,
 id Seq-id , -- WARNING: this used to be optional
 fuzz-from Int-fuzz OPTIONAL ,
 fuzz-to Int-fuzz OPTIONAL }

Packed-seqint ::= SEQUENCE OF Seq-interval

Seq-point ::= SEQUENCE {
 point INTEGER ,
 strand Na-strand OPTIONAL ,
 id Seq-id , -- WARNING: this used to be optional
 fuzz Int-fuzz OPTIONAL }

Packed-seqpnt ::= SEQUENCE {
 strand Na-strand OPTIONAL ,
 id Seq-id ,
 fuzz Int-fuzz OPTIONAL ,
 points SEQUENCE OF INTEGER }

Na-strand ::= ENUMERATED { -- strand of nucleic acid
 unknown (0) ,
 plus (1) ,
 minus (2) ,
 both (3) , -- in forward orientation
 both-rev (4) , -- in reverse orientation
 other (255) }

Seq-bond ::= SEQUENCE { -- bond between residues
 a Seq-point , -- connection to a least one residue
 b Seq-point OPTIONAL } -- other end may not be available

Seq-loc-mix ::= SEQUENCE OF Seq-loc -- this will hold anything

Seq-loc-equiv ::= SET OF Seq-loc -- for a set of equivalent locations

END
```

ASN.1 Specification: seqfeat.asn

See also the online-version of this specification, which may be more up-to-date.

```
--$Revision$
--**********************************************************************
--
-- NCBI Sequence Feature elements
-- by James Ostell, 1990
-- Version 3.0 - June 1994
--
--**********************************************************************

NCBI-Seqfeat DEFINITIONS ::=
BEGIN

EXPORTS Seq-feat, Feat-id, Genetic-code;

IMPORTS Gene-ref FROM NCBI-Gene
 Prot-ref FROM NCBI-Protein
 Org-ref FROM NCBI-Organism
 Variation-ref FROM NCBI-Variation
 BioSource FROM NCBI-BioSource
 RNA-ref FROM NCBI-RNA
 Seq-loc, Giimport-id FROM NCBI-Seqloc
 Pubdesc, Numbering, Heterogen FROM NCBI-Sequence
 Rsite-ref FROM NCBI-Rsite
 Txinit FROM NCBI-TxInit
 Pub-set FROM NCBI-Pub
 Object-id, Dbtag, User-object FROM NCBI-General;

--*** Feature identifiers ********************************
--*

Feat-id ::= CHOICE {
 gibb INTEGER , -- geninfo backbone
 giim Giimport-id , -- geninfo import
 local Object-id , -- for local software use
 general Dbtag } -- for use by various databases

--*** Seq-feat *******************************************
--* sequence feature generalization

Seq-feat ::= SEQUENCE {
 id Feat-id OPTIONAL ,
 data SeqFeatData , -- the specific data
 partial BOOLEAN OPTIONAL , -- incomplete in some way?
 except BOOLEAN OPTIONAL , -- something funny about this?
 comment VisibleString OPTIONAL ,
 product Seq-loc OPTIONAL , -- product of process
 location Seq-loc , -- feature made from
 qual SEQUENCE OF Gb-qual OPTIONAL , -- qualifiers
 title VisibleString OPTIONAL , -- for user defined label
 ext User-object OPTIONAL , -- user defined structure extension
 cit Pub-set OPTIONAL , -- citations for this feature
 exp-ev ENUMERATED { -- evidence for existence of feature
 experimental (1) , -- any reasonable experimental check
 not-experimental (2) } OPTIONAL , -- similarity, pattern, etc
 xref SET OF SeqFeatXref OPTIONAL , -- cite other relevant features
 dbxref SET OF Dbtag OPTIONAL , -- support for xref to other databases
 pseudo BOOLEAN OPTIONAL , -- annotated on pseudogene?
 except-text VisibleString OPTIONAL , -- explain if except=TRUE
 ids SET OF Feat-id OPTIONAL , -- set of Ids; will replace 'id' field
 exts SET OF User-object OPTIONAL } -- set of extensions; will replace 'ext'
field

SeqFeatData ::= CHOICE {
 gene Gene-ref ,
 org Org-ref ,
 cdregion Cdregion ,
 prot Prot-ref ,
 rna RNA-ref ,
 pub Pubdesc , -- publication applies to this seq
 seq Seq-loc , -- to annotate origin from another seq
 imp Imp-feat ,
 region VisibleString, -- named region (globin locus)
 comment NULL , -- just a comment
 bond ENUMERATED {
 disulfide (1) ,
 thiolester (2) ,
 xlink (3) ,
 thioether (4) ,
 other (255) } ,
 site ENUMERATED {
 active (1) ,
 binding (2) ,
 cleavage (3) ,
 inhibit (4) ,
 modified (5),
 glycosylation (6) ,
 myristoylation (7) ,
 mutagenized (8) ,
 metal-binding (9) ,
 phosphorylation (10) ,
 acetylation (11) ,
 amidation (12) ,
 methylation (13) ,
 hydroxylation (14) ,
 sulfatation (15) ,
 oxidative-deamination (16) ,
 pyrrolidone-carboxylic-acid (17) ,
 gamma-carboxyglutamic-acid (18) ,
 blocked (19) ,
 lipid-binding (20) ,
 np-binding (21) ,
 dna-binding (22) ,
 signal-peptide (23) ,
 transit-peptide (24) ,
 transmembrane-region (25) ,
 nitrosylation (26) ,
 other (255) } ,
 rsite Rsite-ref , -- restriction site (for maps really)
 user User-object , -- user defined structure
 txinit Txinit , -- transcription initiation
 num Numbering , -- a numbering system
 psec-str ENUMERATED { -- protein secondary structure
 helix (1) , -- any helix
 sheet (2) , -- beta sheet
 turn (3) } , -- beta or gamma turn
 non-std-residue VisibleString , -- non-standard residue here in seq
 het Heterogen , -- cofactor, prosthetic grp, etc, bound to seq
 biosrc BioSource,
 clone Clone-ref,
 variation Variation-ref
}

SeqFeatXref ::= SEQUENCE { -- both optional because can have one or both
 id Feat-id OPTIONAL , -- the feature copied
 data SeqFeatData OPTIONAL } -- the specific data

--*** CdRegion ***********************************************
--*
--* Instructions to translate from a nucleic acid to a peptide
--* conflict means it's supposed to translate but doesn't
--*


Cdregion ::= SEQUENCE {
 orf BOOLEAN OPTIONAL , -- just an ORF ?
 frame ENUMERATED {
```

```
 not-set (0) , -- not set, code uses one
 one (1) ,
 two (2) ,
 three (3) } DEFAULT not-set , -- reading frame
 conflict BOOLEAN OPTIONAL , -- conflict
 gaps INTEGER OPTIONAL , -- number of gaps on conflict/except
 mismatch INTEGER OPTIONAL , -- number of mismatches on above
 code Genetic-code OPTIONAL , -- genetic code used
 code-break SEQUENCE OF Code-break OPTIONAL , -- individual exceptions
 stops INTEGER OPTIONAL } -- number of stop codons on above

 -- each code is 64 cells long, in the order where
 -- T=0,C=1,A=2,G=3, TTT=0, TTC=1, TCA=4, etc
 -- NOTE: this order does NOT correspond to a Seq-data
 -- encoding. It is "natural" to codon usage instead.
 -- the value in each cell is the AA coded for
 -- start= AA coded only if first in peptide
 -- in start array, if codon is not a legitimate start
 -- codon, that cell will have the "gap" symbol for
 -- that alphabet. Otherwise it will have the AA
 -- encoded when that codon is used at the start.

Genetic-code ::= SET OF CHOICE {
 name VisibleString , -- name of a code
 id INTEGER , -- id in dbase
 ncbieaa VisibleString , -- indexed to IUPAC extended
 ncbi8aa OCTET STRING , -- indexed to NCBI8aa
 ncbistdaa OCTET STRING , -- indexed to NCBIstdaa
 sncbieaa VisibleString , -- start, indexed to IUPAC extended
 sncbi8aa OCTET STRING , -- start, indexed to NCBI8aa
 sncbistdaa OCTET STRING } -- start, indexed to NCBIstdaa

Code-break ::= SEQUENCE { -- specific codon exceptions
 loc Seq-loc , -- location of exception
 aa CHOICE { -- the amino acid
 ncbieaa INTEGER , -- ASCII value of NCBIeaa code
 ncbi8aa INTEGER , -- NCBI8aa code
 ncbistdaa INTEGER } } -- NCBIstdaa code

Genetic-code-table ::= SET OF Genetic-code -- table of genetic codes

--*** Import ***********************************************
--*
--* Features imported from other databases
--*

Imp-feat ::= SEQUENCE {
 key VisibleString ,
 loc VisibleString OPTIONAL , -- original location string
 descr VisibleString OPTIONAL } -- text description
```

```
Gb-qual ::= SEQUENCE {
 qual VisibleString ,
 val VisibleString }


--*** Clone-ref ***********************************************
--*
--* Specification of clone features
--*

Clone-ref ::= SEQUENCE {
 name VisibleString, -- Official clone symbol
 library VisibleString OPTIONAL, -- Library name

 concordant BOOLEAN DEFAULT FALSE, -- OPTIONAL?
 unique BOOLEAN DEFAULT FALSE, -- OPTIONAL?
 placement-method INTEGER {
 end-seq (0), -- Clone placed by end sequence
 insert-alignment (1), -- Clone placed by insert alignment
 sts (2), -- Clone placed by STS
 fish (3),
 fingerprint (4),
 other (255)
 } OPTIONAL,
 clone-seq Clone-seq-set OPTIONAL
}

Clone-seq-set ::= SET OF Clone-seq


Clone-seq ::= SEQUENCE {
 type INTEGER {
 insert (0),
 end (1),
 other (255)
 },
 confidence INTEGER {
 multiple (0), -- Multiple hits
 na (1), -- Unspecified
 nohit-rep (2), -- No hits, repetitive
 nohitnorep (3), -- No hits, not repetitive
 other-chrm (4), -- Hit on different chromosome
 unique (5),
 virtual (6), -- Virtual (hasn't been sequenced)
 other (255)
 } OPTIONAL,
 location Seq-loc, -- location on sequence
 seq Seq-loc OPTIONAL, -- clone sequence location
 align-id Dbtag OPTIONAL
}
```

```
END


--*** Variation-ref ***********************************************
--*
--* Specification of variation features
--*

NCBI-Variation DEFINITIONS ::=
BEGIN

EXPORTS Variation-ref, Variation-inst;

IMPORTS Int-fuzz, User-object, Object-id, Dbtag FROM NCBI-General
 Seq-literal FROM NCBI-Sequence
 Seq-loc FROM NCBI-Seqloc
 Pub FROM NCBI-Pub;

-- --------------------------------------------------------------------------
-- Historically, the dbSNP definitions document data structures used in the
-- processing and annotation of variations by the dbSNP group. The intention
-- is to provide information to clients that reflect internal information
-- produced during the mapping of SNPs
-- --------------------------------------------------------------------------

VariantProperties ::= SEQUENCE {
 version INTEGER,

 -- NOTE:
 -- The format for each of these values is as an integer
 -- Unless otherwise noted, these integers represent a bitwise OR of the
 -- possible values, and as such, these values represent the specific bit
 -- flags that may be set for each of the possible attributes here.

 resource-link INTEGER {
 precious (1), -- Clinical, Pubmed, Cited, (0x01)
 provisional (2), -- Provisional Third Party Annotations (0x02)
 has3D (4), -- Has 3D strcture SNP3D table (0x04)
 submitterLinkout (8), -- SNP->SubSNP->Batch link_out (0x08)
 clinical (16), -- Clinical if LSDB, OMIM, TPA, Diagnostic
 genotypeKit (32) -- Marker exists on high density genotyping kit
 } OPTIONAL,

 gene-function INTEGER {
 no-change (0), -- known to cause no functional changes
 -- since 0 does not combine with any other bit
 -- value, 'no-change' specifically implies that
 -- there are no consequences
 in-gene (1), -- Sequence intervals covered by a gene ID but not
 -- having an aligned transcript (0x01)
 in-gene-5 (2), -- In Gene near 5' (0x02)
```

```
in-gene-3 (4), -- In Gene near 3' (0x04)
intron (8), -- In Intron (0x08)
donor (16), -- In donor splice-site (0x10)
acceptor (32), -- In acceptor splice-site (0x20)
utr-5 (64), -- In 5' UTR (0x40)
utr-3 (128), -- In 3' UTR (0x80)
synonymous (256), -- one allele in the set does not change the encoded
-- amino acid (0x100)
nonsense (512), -- one allele in the set changes to STOP codon
-- (TER). (0x200)
missense (1024), -- one allele in the set changes protein peptide
-- (0x400)
frameshift (2048), -- one allele in the set changes all downstream
-- amino acids (0x800)

in-start-codon(4096), -- the variant is observed in a start codon (0x1000)
up-regulator(8192), -- the variant causes increased transcription
-- (0x2000)
down-regulator(16384) -- the variant causes decreased transcription
-- (0x4000)
} OPTIONAL,

mapping INTEGER {
has-other-snp (1), -- Another SNP has the same mapped positions
-- on reference assembly (0x01)
has-assembly-conflict (2), -- Weight 1 or 2 SNPs that map to different
-- chromosomes on different assemblies (0x02)
is-assembly-specific (4) -- Only maps to 1 assembly (0x04)
} OPTIONAL,

-- This is *NOT* a bitfield
weight INTEGER {
is-uniquely-placed(1),
placed-twice-on-same-chrom(2),
placed-twice-on-diff-chrom(3),
many-placements(10)
} OPTIONAL,

allele-freq INTEGER {
is-mutation (1), -- low frequency variation that is cited in journal
-- and other reputable sources (0x01)
above-5pct-all (2), -- >5% minor allele freq in each and all
-- populations (0x02)
above-5pct-1plus (4), -- >5% minor allele freq in 1+ populations (0x04)
validated (8) -- Bit is set if the variant has 2+ minor allele
-- count based on freq or genotype data
} OPTIONAL,

genotype INTEGER {
in-haplotype-set (1), -- Exists in a haplotype tagging set (0x01)
has-genotypes (2) -- SNP has individual genotype (0x02)
```

```
 } OPTIONAL,

 hapmap INTEGER {
 phase1-genotyped (1), -- Phase 1 genotyped; filtered, non-redundant
 -- (0x01)
 phase2-genotyped (2), -- Phase 2 genotyped; filtered, non-redundant
 -- (0x02)
 phase3-genotyped (4) -- Phase 3 genotyped; filtered, non-redundant
 -- (0x04)
 } OPTIONAL,

 quality-check INTEGER {
 contig-allele-missing (1), -- Reference sequence allele at the mapped
 -- position is not present in the SNP
 -- allele list, adjusted for orientation
 -- (0x01)
 withdrawn-by-submitter (2), -- One member SS is withdrawn by submitter
 -- (0x02)
 non-overlapping-alleles (4), -- RS set has 2+ alleles from different
 -- submissions and these sets share no
 -- alleles in common (0x04)
 strain-specific (8), -- Straing specific fixed difference (0x08)
 genotype-conflict (16) -- Has Genotype Conflict (0x10)
 } OPTIONAL
}

Phenotype ::= SEQUENCE {
 source VisibleString OPTIONAL,
 term VisibleString OPTIONAL,
 xref SET OF Dbtag OPTIONAL,

 -- does this variant have known clinical significance?
 clinical-significance INTEGER {
 unknown (0),
 untested (1),
 non-pathogenic (2),
 probable-non-pathogenic (3),
 probable-pathogenic (4),
 pathogenic (5),
 other (255)
 } OPTIONAL
}

Population-data ::= SEQUENCE {
 -- assayed population (e.g. HAPMAP-CEU)
 population VisibleString,
 genotype-frequency REAL OPTIONAL,
 chromosomes-tested INTEGER OPTIONAL,
 sample-ids SET OF Object-id OPTIONAL
}
```

```
Ext-loc ::= SEQUENCE {
 id Object-id,
 location Seq-loc
}

Variation-ref ::= SEQUENCE {
 -- ids (i.e., SNP rsid / ssid, dbVar nsv/nssv)
 -- expected values include 'dbSNP|rs12334', 'dbSNP|ss12345', 'dbVar|nsv1'
 --
 -- we relate three kinds of IDs here:
 -- - our current object's id
 -- - the id of this object's parent, if it exists
 -- - the sample ID that this item originates from
 id Dbtag OPTIONAL,
 parent-id Dbtag OPTIONAL,
 sample-id Object-id OPTIONAL,
 other-ids SET OF Dbtag OPTIONAL,

 -- names and synonyms
 -- some variants have well-known canonical names and possible accepted
 -- synonyms
 name VisibleString OPTIONAL,
 synonyms SET OF VisibleString OPTIONAL,

 -- tag for comment and descriptions
 description VisibleString OPTIONAL,

 -- phenotype
 phenotype SET OF Phenotype OPTIONAL,

 -- sequencing / acuisition method
 method SET OF INTEGER {
 unknown (0),
 bac-acgh (1),
 computational (2),
 curated (3),
 digital-array (4),
 expression-array (5),
 fish (6),
 flanking-sequence (7),
 maph (8),
 mcd-analysis (9),
 mlpa (10),
 oea-assembly (11),
 oligo-acgh (12),
 paired-end (13),
 pcr (14),
 qpcr (15),
 read-depth (16),
 roma (17),
 rt-pcr (18),
 sage (19),
 sequence-alignment (20),
 sequencing (21),
 snp-array (22),
 snp-genoytyping (23),
 southern (24),
 western (25),

 other (255)
 } OPTIONAL,

 -- Note about SNP representation and pretinent fields: allele-frequency,
 -- population, quality-codes:
 -- The case of multiple alleles for a SNP would be described by
 -- parent-feature of type Variation-set.diff-alleles, where the child
 -- features of type Variation-inst, all at the same location, would
 -- describe individual alleles.

 -- population data
 population-data SET OF Population-data OPTIONAL,

 -- variant properties bit fields
 variant-prop VariantProperties OPTIONAL,

 -- has this variant been validated?
 validated BOOLEAN OPTIONAL,

 -- link-outs to GeneTests database
 clinical-test SET OF Dbtag OPTIONAL,

 -- origin of this allele, if known
 allele-origin INTEGER {
 unknown (0),
 germline (1),
 somatic (2),
 inherited (3),
 paternal (4),
 maternal (5),
 de-novo (6),
 biparental (7),
 uniparental (8),
 not-tested (9),
 tested-inconclusive (10),

 other (255)
 } OPTIONAL,

 -- observed allele state, if known
 allele-state INTEGER {
 unknown (0),
 homozygous (1),
 heterozygous (2),
 hemizygous (3),
 nullizygous (4),
 other (255)
 } OPTIONAL,

 allele-frequency REAL OPTIONAL,

 -- is this variant the ancestral allele?
 is-ancestral-allele BOOLEAN OPTIONAL,

 -- publication support.
 -- Note: made this pub instead of pub-equiv, since
 -- Pub can be pub-equiv and pub-equiv is a set of pubs, but it looks like
 -- Pub is more often used as top-level container
 pub Pub OPTIONAL,

 data CHOICE {
 unknown NULL,
 note VisibleString, --free-form
 uniparental-disomy NULL,

 -- actual sequence-edit at feat.location
 instance Variation-inst,

 -- Set of related Variations.
 -- Location of the set equals to the union of member locations
 set SEQUENCE {
 type INTEGER {
 unknown (0),
 compound (1), -- complex change at the same location on the
 -- same molecule
 products (2), -- different products arising from the same
 -- variation in a precursor, e.g. r.[13g>a,
 -- 13_88del]
 haplotype (3), -- changes on the same allele, e.g
 -- r.[13g>a;15u>c]
 alleles (4), -- changes on different alleles in the same
 -- genotype, e.g. g.[476C>T]+[476C>T]
 mosaic (5), -- different genotypes in the same individual
 individual (6), -- same organism; allele relationship unknown,
 -- e.g. g.[476C>T(+)183G>C]
 population (7), -- population
 other (255)
 },
 variations SET OF Variation-ref,
 name VisibleString OPTIONAL
 }
 },

 consequence SET OF CHOICE {
 unknown NULL,
 splicing NULL, --some effect on splicing
 note VisibleString, --freeform

 -- Describe resulting variation in the product, e.g. missense,
 -- nonsense, silent, neutral, etc in a protein, that arises from
 -- THIS variation.
 variation Variation-ref,

 -- see http://www.hgvs.org/mutnomen/recs-prot.html
 frameshift SEQUENCE {
 phase INTEGER OPTIONAL,
 x-length INTEGER OPTIONAL
 },

 loss-of-heterozygosity SEQUENCE {
 -- In germline comparison, it will be reference genome assembly
 -- (default) or reference/normal population. In somatic mutation,
 -- it will be a name of the normal tissue.
 reference VisibleString OPTIONAL,

 -- Name of the testing subject type or the testing tissue.
 test VisibleString OPTIONAL
 }
 } OPTIONAL,

 -- Observed location, if different from the parent set or feature.location.
 location Seq-loc OPTIONAL,

 -- reference other locs, e.g. mapped source
 ext-locs SET OF Ext-loc OPTIONAL,

 ext User-object OPTIONAL

}

Delta-item ::= SEQUENCE {
 seq CHOICE {
 literal Seq-literal,
 loc Seq-loc,
 this NULL --same location as variation-ref itself
 },

 -- Multiplier allows representing a tandem, e.g. ATATAT as AT*3
 -- This allows describing CNV/SSR where delta=self with a
 -- multiplier which specifies the count of the repeat unit.

 multiplier INTEGER OPTIONAL, --assumed 1 if not specified.
 multiplier-fuzz Int-fuzz OPTIONAL
}
```

```
-- Variation instance
Variation-inst ::= SEQUENCE {
 type INTEGER {
 unknown (0),
 identity (1), -- delta = this
 inv (2), -- inversion: delta =
 -- reverse-comp(feat.location)
 snp (3), -- delins where len(del) = len(ins) = 1
 mnp (4), -- delins where len(del) = len(ins) > 1
 delins (5), -- delins where len(del) != len(ins)
 del (6), -- deltaseq is empty
 ins (7), -- deltaseq contains [this, ins] or [ins, this]
 microsatellite (8), -- location describes tandem sequence;
 -- delta is the repeat-unit with a multiplier
 transposon (9), -- delta refers to equivalent sequence in
 -- another location.
 -- ext-loc describes donor sequence, if known
 -- (could be location itself)
 cnv (10), -- general CNV class, indicating "local"
 -- rearrangement

 -- Below are four possible copy configurations,
 -- where delta is a seq-loc on the same sequence.
 -- If the repeat is represented on the sequence, it is
 -- described like a transposon; if it is a de-novo
 -- repeat, it is described like an insertion.
 direct-copy (11), -- delta sequence is located upstream, same
 -- strand
 rev-direct-copy (12), -- delta sequence is downstream, same strand
 inverted-copy (13), -- delta sequence is upstream, opposite strand
 everted-copy (14), -- delta sequence is downstream, opposite strand

 translocation (15), -- feat.location is swapped with delta (i.e.
 -- reciprocal transposon)
 prot-missense (16),
 prot-nonsense (17),
 prot-neutral (18),
 prot-silent (19),
 prot-other (20),

 other (255)
 },

 -- Sequence that replaces the location, in biological order.
 delta SEQUENCE OF Delta-item
}

END


--**********************************************************************
```

```
--
-- NCBI Restriction Sites
-- by James Ostell, 1990
-- version 0.8
--
--**********************************************************************

NCBI-Rsite DEFINITIONS ::=
BEGIN

EXPORTS Rsite-ref;

IMPORTS Dbtag FROM NCBI-General;

Rsite-ref ::= CHOICE {
 str VisibleString , -- may be unparsable
 db Dbtag } -- pointer to a restriction site database

END

--**********************************************************************
--
-- NCBI RNAs
-- by James Ostell, 1990
-- version 0.8
--
--**********************************************************************

NCBI-RNA DEFINITIONS ::=
BEGIN

EXPORTS RNA-ref, Trna-ext, RNA-gen, RNA-qual, RNA-qual-set;

IMPORTS Seq-loc FROM NCBI-Seqloc;

--*** rnas ***********************************************
--*
--* various rnas
--*
 -- minimal RNA sequence
RNA-ref ::= SEQUENCE {
 type ENUMERATED { -- type of RNA feature
 unknown (0) ,
 premsg (1) ,
 mRNA (2) ,
 tRNA (3) ,
 rRNA (4) ,
 snRNA (5) , -- will become ncRNA, with RNA-gen.class = snRNA
 scRNA (6) , -- will become ncRNA, with RNA-gen.class = scRNA
 snoRNA (7) , -- will become ncRNA, with RNA-gen.class = snoRNA
 ncRNA (8) , -- non-coding RNA; subsumes snRNA, scRNA, snoRNA
 tmRNA (9) ,
 miscRNA (10) ,
 other (255) } ,
 pseudo BOOLEAN OPTIONAL ,
 ext CHOICE {
 name VisibleString , -- for naming "other" type
 tRNA Trna-ext , -- for tRNAs
 gen RNA-gen } OPTIONAL -- generic fields for ncRNA, tmRNA, miscRNA
 }

Trna-ext ::= SEQUENCE { -- tRNA feature extensions
 aa CHOICE { -- aa this carries
 iupacaa INTEGER ,
 ncbieaa INTEGER ,
 ncbi8aa INTEGER ,
 ncbistdaa INTEGER } OPTIONAL ,
 codon SET OF INTEGER OPTIONAL , -- codon(s) as in Genetic-code
 anticodon Seq-loc OPTIONAL } -- location of anticodon

RNA-gen ::= SEQUENCE {
 class VisibleString OPTIONAL , -- for ncRNAs, the class of non-coding RNA:
 -- examples: antisense_RNA, guide_RNA, snRNA
 product VisibleString OPTIONAL ,
 quals RNA-qual-set OPTIONAL -- e.g., tag_peptide qualifier for tmRNAs
}

RNA-qual ::= SEQUENCE { -- Additional data values for RNA-gen,
 qual VisibleString , -- in a tag (qual), value (val) format
 val VisibleString }

RNA-qual-set ::= SEQUENCE OF RNA-qual

END

--**********************************************************************
--
-- NCBI Genes
-- by James Ostell, 1990
-- version 0.8
--
--**********************************************************************

NCBI-Gene DEFINITIONS ::=
BEGIN

EXPORTS Gene-ref, Gene-nomenclature;

IMPORTS Dbtag FROM NCBI-General;

--*** Gene ***********************************************
--*
```

```
--* reference to a gene
--*

Gene-ref ::= SEQUENCE {
 locus VisibleString OPTIONAL , -- Official gene symbol
 allele VisibleString OPTIONAL , -- Official allele designation
 desc VisibleString OPTIONAL , -- descriptive name
 maploc VisibleString OPTIONAL , -- descriptive map location
 pseudo BOOLEAN DEFAULT FALSE , -- pseudogene
 db SET OF Dbtag OPTIONAL , -- ids in other dbases
 syn SET OF VisibleString OPTIONAL , -- synonyms for locus
 locus-tag VisibleString OPTIONAL , -- systematic gene name (e.g., MI0001,
ORF0069)
 formal-name Gene-nomenclature OPTIONAL
}

Gene-nomenclature ::= SEQUENCE {
 status ENUMERATED {
 unknown (0) ,
 official (1) ,
 interim (2)
 } ,
 symbol VisibleString OPTIONAL ,
 name VisibleString OPTIONAL ,
 source Dbtag OPTIONAL
}

END


--**********************************************************************
--
-- NCBI Organism
-- by James Ostell, 1994
-- version 3.0
--
--**********************************************************************

NCBI-Organism DEFINITIONS ::=
BEGIN

EXPORTS Org-ref;

IMPORTS Dbtag FROM NCBI-General;

--*** Org-ref ***********************************************
--*
--* Reference to an organism
--* defines only the organism.. lower levels of detail for biological
--* molecules are provided by the Source object
--*
```

```
Org-ref ::= SEQUENCE {
 taxname VisibleString OPTIONAL , -- preferred formal name
 common VisibleString OPTIONAL , -- common name
 mod SET OF VisibleString OPTIONAL , -- unstructured modifiers
 db SET OF Dbtag OPTIONAL , -- ids in taxonomic or culture dbases
 syn SET OF VisibleString OPTIONAL , -- synonyms for taxname or common
 orgname OrgName OPTIONAL }


OrgName ::= SEQUENCE {
 name CHOICE {
 binomial BinomialOrgName , -- genus/species type name
 virus VisibleString , -- virus names are different
 hybrid MultiOrgName , -- hybrid between organisms
 namedhybrid BinomialOrgName , -- some hybrids have genus x species name
 partial PartialOrgName } OPTIONAL , -- when genus not known
 attrib VisibleString OPTIONAL , -- attribution of name
 mod SEQUENCE OF OrgMod OPTIONAL ,
 lineage VisibleString OPTIONAL , -- lineage with semicolon separators
 gcode INTEGER OPTIONAL , -- genetic code (see CdRegion)
 mgcode INTEGER OPTIONAL , -- mitochondrial genetic code
 div VisibleString OPTIONAL } -- GenBank division code


OrgMod ::= SEQUENCE {
 subtype INTEGER {
 strain (2) ,
 substrain (3) ,
 type (4) ,
 subtype (5) ,
 variety (6) ,
 serotype (7) ,
 serogroup (8) ,
 serovar (9) ,
 cultivar (10) ,
 pathovar (11) ,
 chemovar (12) ,
 biovar (13) ,
 biotype (14) ,
 group (15) ,
 subgroup (16) ,
 isolate (17) ,
 common (18) ,
 acronym (19) ,
 dosage (20) , -- chromosome dosage of hybrid
 nat-host (21) , -- natural host of this specimen
 sub-species (22) ,
 specimen-voucher (23) ,
 authority (24) ,
 forma (25) ,
 forma-specialis (26) ,
 ecotype (27) ,
 synonym (28) ,
 anamorph (29) ,
 teleomorph (30) ,
 breed (31) ,
 gb-acronym (32) , -- used by taxonomy database
 gb-anamorph (33) , -- used by taxonomy database
 gb-synonym (34) , -- used by taxonomy database
 culture-collection (35) ,
 bio-material (36) ,
 metagenome-source (37) ,
 old-lineage (253) ,
 old-name (254) ,
 other (255) } , -- ASN5: old-name (254) will be added to next spec
 subname VisibleString ,
 attrib VisibleString OPTIONAL } -- attribution/source of name

BinomialOrgName ::= SEQUENCE {
 genus VisibleString , -- required
 species VisibleString OPTIONAL , -- species required if subspecies used
 subspecies VisibleString OPTIONAL }

MultiOrgName ::= SEQUENCE OF OrgName -- the first will be used to assign
division

PartialOrgName ::= SEQUENCE OF TaxElement -- when we don't know the genus

TaxElement ::= SEQUENCE {
 fixed-level INTEGER {
 other (0) , -- level must be set in string
 family (1) ,
 order (2) ,
 class (3) } ,
 level VisibleString OPTIONAL ,
 name VisibleString }

END


--**********************************************************************
--
-- NCBI BioSource
-- by James Ostell, 1994
-- version 3.0
--
--**********************************************************************

NCBI-BioSource DEFINITIONS ::=
BEGIN
```

```
EXPORTS BioSource;

IMPORTS Org-ref FROM NCBI-Organism;

--*********************************************************************
--
-- BioSource gives the source of the biological material
-- for sequences
--
--*********************************************************************

BioSource ::= SEQUENCE {
 genome INTEGER { -- biological context
 unknown (0) ,
 genomic (1) ,
 chloroplast (2) ,
 chromoplast (3) ,
 kinetoplast (4) ,
 mitochondrion (5) ,
 plastid (6) ,
 macronuclear (7) ,
 extrachrom (8) ,
 plasmid (9) ,
 transposon (10) ,
 insertion-seq (11) ,
 cyanelle (12) ,
 proviral (13) ,
 virion (14) ,
 nucleomorph (15) ,
 apicoplast (16) ,
 leucoplast (17) ,
 proplastid (18) ,
 endogenous-virus (19) ,
 hydrogenosome (20) ,
 chromosome (21) ,
 chromatophore (22)
 } DEFAULT unknown ,
 origin INTEGER {
 unknown (0) ,
 natural (1) , -- normal biological entity
 natmut (2) , -- naturally occurring mutant
 mut (3) , -- artificially mutagenized
 artificial (4) , -- artificially engineered
 synthetic (5) , -- purely synthetic
 other (255)
 } DEFAULT unknown ,
 org Org-ref ,
 subtype SEQUENCE OF SubSource OPTIONAL ,
 is-focus NULL OPTIONAL , -- to distinguish biological focus
 pcr-primers PCRReactionSet OPTIONAL }
```

```
PCRReactionSet ::= SET OF PCRReaction

PCRReaction ::= SEQUENCE {
 forward PCRPrimerSet OPTIONAL ,
 reverse PCRPrimerSet OPTIONAL }

PCRPrimerSet ::= SET OF PCRPrimer

PCRPrimer ::= SEQUENCE {
 seq PCRPrimerSeq OPTIONAL ,
 name PCRPrimerName OPTIONAL }

PCRPrimerSeq ::= VisibleString

PCRPrimerName ::= VisibleString

SubSource ::= SEQUENCE {
 subtype INTEGER {
 chromosome (1) ,
 map (2) ,
 clone (3) ,
 subclone (4) ,
 haplotype (5) ,
 genotype (6) ,
 sex (7) ,
 cell-line (8) ,
 cell-type (9) ,
 tissue-type (10) ,
 clone-lib (11) ,
 dev-stage (12) ,
 frequency (13) ,
 germline (14) ,
 rearranged (15) ,
 lab-host (16) ,
 pop-variant (17) ,
 tissue-lib (18) ,
 plasmid-name (19) ,
 transposon-name (20) ,
 insertion-seq-name (21) ,
 plastid-name (22) ,
 country (23) ,
 segment (24) ,
 endogenous-virus-name (25) ,
 transgenic (26) ,
 environmental-sample (27) ,
 isolation-source (28) ,
 lat-lon (29) , -- +/- decimal degrees
 collection-date (30) , -- DD-MMM-YYYY format
 collected-by (31) , -- name of person who collected the sample
 identified-by (32) , -- name of person who identified the sample
 fwd-primer-seq (33) , -- sequence (possibly more than one; semicolon-
separated)
 rev-primer-seq (34) , -- sequence (possibly more than one; semicolon-
separated)
 fwd-primer-name (35) ,
 rev-primer-name (36) ,
 metagenomic (37) ,
 mating-type (38) ,
 linkage-group (39) ,
 haplogroup (40) ,
 other (255) } ,
 name VisibleString ,
 attrib VisibleString OPTIONAL } -- attribution/source of this name

END

--**********************************************************************
--
-- NCBI Protein
-- by James Ostell, 1990
-- version 0.8
--
--**********************************************************************

NCBI-Protein DEFINITIONS ::=
BEGIN

EXPORTS Prot-ref;

IMPORTS Dbtag FROM NCBI-General;

--*** Prot-ref ***********************************************
--*
--* Reference to a protein name
--*

Prot-ref ::= SEQUENCE {
 name SET OF VisibleString OPTIONAL , -- protein name
 desc VisibleString OPTIONAL , -- description (instead of name)
 ec SET OF VisibleString OPTIONAL , -- E.C. number(s)
 activity SET OF VisibleString OPTIONAL , -- activities
 db SET OF Dbtag OPTIONAL , -- ids in other dbases
 processed ENUMERATED { -- processing status
 not-set (0) ,
 preprotein (1) ,
 mature (2) ,
 signal-peptide (3) ,
 transit-peptide (4) } DEFAULT not-set }

END
--********************************************************************
--
```

```
-- Transcription Initiation Site Feature Data Block
-- James Ostell, 1991
-- Philip Bucher, David Ghosh
-- version 1.1
--
--
--
--*********************************************************************

NCBI-TxInit DEFINITIONS ::=
BEGIN

EXPORTS Txinit;

IMPORTS Gene-ref FROM NCBI-Gene
 Prot-ref FROM NCBI-Protein
 Org-ref FROM NCBI-Organism;

Txinit ::= SEQUENCE {
 name VisibleString , -- descriptive name of initiation site
 syn SEQUENCE OF VisibleString OPTIONAL , -- synonyms
 gene SEQUENCE OF Gene-ref OPTIONAL , -- gene(s) transcribed
 protein SEQUENCE OF Prot-ref OPTIONAL , -- protein(s) produced
 rna SEQUENCE OF VisibleString OPTIONAL , -- rna(s) produced
 expression VisibleString OPTIONAL , -- tissue/time of expression
 txsystem ENUMERATED { -- transcription apparatus used at this site
 unknown (0) ,
 pol1 (1) , -- eukaryotic Pol I
 pol2 (2) , -- eukaryotic Pol II
 pol3 (3) , -- eukaryotic Pol III
 bacterial (4) ,
 viral (5) ,
 rna (6) , -- RNA replicase
 organelle (7) ,
 other (255) } ,
 txdescr VisibleString OPTIONAL , -- modifiers on txsystem
 txorg Org-ref OPTIONAL , -- organism supplying transcription apparatus
 mapping-precise BOOLEAN DEFAULT FALSE , -- mapping precise or approx
 location-accurate BOOLEAN DEFAULT FALSE , -- does Seq-loc reflect mapping
 inittype ENUMERATED {
 unknown (0) ,
 single (1) ,
 multiple (2) ,
 region (3) } OPTIONAL ,
 evidence SET OF Tx-evidence OPTIONAL }

Tx-evidence ::= SEQUENCE {
 exp-code ENUMERATED {
 unknown (0) ,
 rna-seq (1) , -- direct RNA sequencing
 rna-size (2) , -- RNA length measurement
 np-map (3) , -- nuclease protection mapping with homologous sequence ladder
 np-size (4) , -- nuclease protected fragment length measurement
 pe-seq (5) , -- dideoxy RNA sequencing
 cDNA-seq (6) , -- full-length cDNA sequencing
 pe-map (7) , -- primer extension mapping with homologous sequence ladder
 pe-size (8) , -- primer extension product length measurement
 pseudo-seq (9) , -- full-length processed pseudogene sequencing
 rev-pe-map (10) , -- see NOTE (1) below
 other (255) } ,
 expression-system ENUMERATED {
 unknown (0) ,
 physiological (1) ,
 in-vitro (2) ,
 oocyte (3) ,
 transfection (4) ,
 transgenic (5) ,
 other (255) } DEFAULT physiological ,
 low-prec-data BOOLEAN DEFAULT FALSE ,
 from-homolog BOOLEAN DEFAULT FALSE } -- experiment actually done on
 -- close homolog

 -- NOTE (1) length measurement of a reverse direction primer-extension
 -- product (blocked by RNA 5'end) by comparison with
 -- homologous sequence ladder (J. Mol. Biol. 199, 587)

END
```

ASN.1 Specification: seqalign.asn

See also the online-version of this specification, which may be more up-to-date.

```
--$Revision$
--**********************************************************************
--
-- NCBI Sequence Alignment elements
-- by James Ostell, 1990
--
--**********************************************************************

NCBI-Seqalign DEFINITIONS ::=
BEGIN

EXPORTS Seq-align, Score, Score-set, Seq-align-set;

IMPORTS Seq-id, Seq-loc , Na-strand FROM NCBI-Seqloc
 User-object, Object-id FROM NCBI-General;

--*** Sequence Alignment ********************************
--*

Seq-align-set ::= SET OF Seq-align
```

```
Seq-align ::= SEQUENCE {
 type ENUMERATED {
 not-set (0) ,
 global (1) ,
 diags (2) , -- unbroken, but not ordered, diagonals
 partial (3) , -- mapping pieces together
 disc (4) , -- discontinuous alignment
 other (255) } ,
 dim INTEGER OPTIONAL , -- dimensionality
 score SET OF Score OPTIONAL , -- for whole alignment
 segs CHOICE { -- alignment data
 dendiag SEQUENCE OF Dense-diag ,
 denseg Dense-seg ,
 std SEQUENCE OF Std-seg ,
 packed Packed-seg ,
 disc Seq-align-set,
 spliced Spliced-seg,
 sparse Sparse-seg
 } ,

 -- regions of sequence over which align
 -- was computed
 bounds SET OF Seq-loc OPTIONAL,

 -- alignment id
 id SEQUENCE OF Object-id OPTIONAL,

 --extra info
 ext SEQUENCE OF User-object OPTIONAL
}

Dense-diag ::= SEQUENCE { -- for (multiway) diagonals
 dim INTEGER DEFAULT 2 , -- dimensionality
 ids SEQUENCE OF Seq-id , -- sequences in order
 starts SEQUENCE OF INTEGER , -- start OFFSETS in ids order
 len INTEGER , -- len of aligned segments
 strands SEQUENCE OF Na-strand OPTIONAL ,
 scores SET OF Score OPTIONAL }

 -- Dense-seg: the densist packing for sequence alignments only.
 -- a start of -1 indicates a gap for that sequence of
 -- length lens.
 --
 -- id=100 AAGGCCTTTTAGAGATGATGATGATGATGA
 -- id=200 AAGGCCTTTTAG.......GATGATGATGA
 -- id=300 ....CCTTTTAGAGATGATGAT....ATGA
 --
 -- dim = 3, numseg = 6, ids = { 100, 200, 300 }
 -- starts = { 0,0,-1, 4,4,0, 12,-1,8, 19,12,15, 22,15,-1, 26,19,18 }
 -- lens = { 4, 8, 7, 3, 4, 4 }
 --
```

```
Dense-seg ::= SEQUENCE { -- for (multiway) global or partial alignments
 dim INTEGER DEFAULT 2 , -- dimensionality
 numseg INTEGER , -- number of segments here
 ids SEQUENCE OF Seq-id , -- sequences in order
 starts SEQUENCE OF INTEGER , -- start OFFSETS in ids order within segs
 lens SEQUENCE OF INTEGER , -- lengths in ids order within segs
 strands SEQUENCE OF Na-strand OPTIONAL ,
 scores SEQUENCE OF Score OPTIONAL } -- score for each seg

Packed-seg ::= SEQUENCE { -- for (multiway) global or partial alignments
 dim INTEGER DEFAULT 2 , -- dimensionality
 numseg INTEGER , -- number of segments here
 ids SEQUENCE OF Seq-id , -- sequences in order
 starts SEQUENCE OF INTEGER , -- start OFFSETS in ids order for whole
alignment
 present OCTET STRING , -- Boolean if each sequence present or absent in
 -- each segment
 lens SEQUENCE OF INTEGER , -- length of each segment
 strands SEQUENCE OF Na-strand OPTIONAL ,
 scores SEQUENCE OF Score OPTIONAL } -- score for each segment

Std-seg ::= SEQUENCE {
 dim INTEGER DEFAULT 2 , -- dimensionality
 ids SEQUENCE OF Seq-id OPTIONAL ,
 loc SEQUENCE OF Seq-loc ,
 scores SET OF Score OPTIONAL }


Spliced-seg ::= SEQUENCE {
 -- product is either protein or transcript (cDNA)
 product-id Seq-id OPTIONAL,
 genomic-id Seq-id OPTIONAL,

 -- should be 'plus' or 'minus'
 product-strand Na-strand OPTIONAL ,
 genomic-strand Na-strand OPTIONAL ,

 product-type ENUMERATED {
 transcript(0),
 protein(1)
 },

 -- set of segments involved
 -- each segment corresponds to one exon
 -- exons are always in biological order
 exons SEQUENCE OF Spliced-exon ,

 -- start of poly(A) tail on the transcript
 -- For sense transcripts:
 -- aligned product positions < poly-a <= product-length
 -- poly-a == product-length indicates inferred poly(A) tail at transcript's
end
 -- For antisense transcripts:
 -- -1 <= poly-a < aligned product positions
 -- poly-a == -1 indicates inferred poly(A) tail at transcript's start
 poly-a INTEGER OPTIONAL,

 -- length of the product, in bases/residues
 -- from this (or from poly-a if present), a 3' unaligned length can be
extracted
 product-length INTEGER OPTIONAL,

 -- alignment descriptors / modifiers
 -- this provides us a set for extension
 modifiers SET OF Spliced-seg-modifier OPTIONAL
}

Spliced-seg-modifier ::= CHOICE {
 -- protein aligns from the start and the first codon
 -- on both product and genomic is start codon
 start-codon-found BOOLEAN,

 -- protein aligns to it's end and there is stop codon
 -- on the genomic right after the alignment
 stop-codon-found BOOLEAN
}


-- complete or partial exon
-- two consecutive Spliced-exons may belong to one exon
Spliced-exon ::= SEQUENCE {
 -- product-end >= product-start
 product-start Product-pos ,
 product-end Product-pos ,

 -- genomic-end >= genomic-start
 genomic-start INTEGER ,
 genomic-end INTEGER ,

 -- product is either protein or transcript (cDNA)
 product-id Seq-id OPTIONAL ,
 genomic-id Seq-id OPTIONAL ,

 -- should be 'plus' or 'minus'
 product-strand Na-strand OPTIONAL ,

 -- genomic-strand represents the strand of translation
 genomic-strand Na-strand OPTIONAL ,

 -- basic seqments always are in biologic order
 parts SEQUENCE OF Spliced-exon-chunk OPTIONAL ,
 -- scores for this exon
 scores Score-set OPTIONAL ,

 -- splice sites
 acceptor-before-exon Splice-site OPTIONAL,
 donor-after-exon Splice-site OPTIONAL,

 -- flag: is this exon complete or partial?
 partial BOOLEAN OPTIONAL,

 --extra info
 ext SEQUENCE OF User-object OPTIONAL
}


Product-pos ::= CHOICE {
 nucpos INTEGER,
 protpos Prot-pos
}


-- codon based position on protein (1/3 of aminoacid)
Prot-pos ::= SEQUENCE {
 -- standard protein position
 amin INTEGER ,

 -- 0, 1, 2, or 3 as for Cdregion
 -- 0 = not set
 -- 1, 2, 3 = actual frame
 frame INTEGER DEFAULT 0
}


-- Spliced-exon-chunk: piece of an exon
-- lengths are given in nucleotide bases (1/3 of aminoacid when product is a
-- protein)
Spliced-exon-chunk ::= CHOICE {
 -- both sequences represented, product and genomic sequences match
 match INTEGER ,

 -- both sequences represented, product and genomic sequences do not match
 mismatch INTEGER ,

 -- both sequences are represented, there is sufficient similarity
 -- between product and genomic sequences. Can be used to replace stretches
 -- of matches and mismatches, mostly for protein to genomic where
 -- definition of match or mismatch depends on translation table
 diag INTEGER ,

 -- insertion in product sequence (i.e. gap in the genomic sequence)
 product-ins INTEGER ,

 -- insertion in genomic sequence (i.e. gap in the product sequence)
 genomic-ins INTEGER
}


-- site involved in splice
Splice-site ::= SEQUENCE {
 -- typically two bases in the intronic region, always
 -- in IUPAC format
 bases VisibleString
}


-- ===========================================================================
--
-- Sparse-seg follows the semantics of dense-seg and is more optimal for
-- representing sparse multiple alignments
--
-- ===========================================================================


Sparse-seg ::= SEQUENCE {
 master-id Seq-id OPTIONAL,

 -- pairwise alignments constituting this multiple alignment
 rows SET OF Sparse-align,

 -- per-row scores
 row-scores SET OF Score OPTIONAL,

 -- index of extra items
 ext SET OF Sparse-seg-ext OPTIONAL
}

Sparse-align ::= SEQUENCE {
 first-id Seq-id,
 second-id Seq-id,

 numseg INTEGER, --number of segments
 first-starts SEQUENCE OF INTEGER , --starts on the first sequence [numseg]
 second-starts SEQUENCE OF INTEGER , --starts on the second sequence [numseg]
 lens SEQUENCE OF INTEGER , --lengths of segments [numseg]
 second-strands SEQUENCE OF Na-strand OPTIONAL ,

 -- per-segment scores
 seg-scores SET OF Score OPTIONAL
}

Sparse-seg-ext ::= SEQUENCE {
 --seg-ext SET OF {
 -- index INTEGER,
 -- data User-field
 -- }
 index INTEGER
}



-- use of Score is discouraged for external ASN.1 specifications
Score ::= SEQUENCE {
 id Object-id OPTIONAL ,
 value CHOICE {
 real REAL ,
 int INTEGER
 }
}

-- use of Score-set is encouraged for external ASN.1 specifications
Score-set ::= SET OF Score

END
```

ASN.1 Specification: seqres.asn

See also the online-version of this specification, which may be more up-to-date.

```
--$Revision$
--**********************************************************************
--
-- NCBI Sequence Analysis Results (other than alignments)
-- by James Ostell, 1990
--
--**********************************************************************

NCBI-Seqres DEFINITIONS ::=
BEGIN

EXPORTS Seq-graph;

IMPORTS Seq-loc FROM NCBI-Seqloc;

--*** Sequence Graph ********************************
--*
--* for values mapped by residue or range to sequence
--*

Seq-graph ::= SEQUENCE {
 title VisibleString OPTIONAL ,
 comment VisibleString OPTIONAL ,
 loc Seq-loc , -- region this applies to
 title-x VisibleString OPTIONAL , -- title for x-axis
 title-y VisibleString OPTIONAL ,
 comp INTEGER OPTIONAL , -- compression (residues/value)
 a REAL OPTIONAL , -- for scaling values
 b REAL OPTIONAL , -- display = (a x value) + b
 numval INTEGER , -- number of values in graph
 graph CHOICE {
 real Real-graph ,
 int Int-graph ,
 byte Byte-graph } }

Real-graph ::= SEQUENCE {
 max REAL , -- top of graph
 min REAL , -- bottom of graph
 axis REAL , -- value to draw axis on
 values SEQUENCE OF REAL }

Int-graph ::= SEQUENCE {
 max INTEGER ,
 min INTEGER ,
 axis INTEGER ,
 values SEQUENCE OF INTEGER }

Byte-graph ::= SEQUENCE { -- integer from 0-255
 max INTEGER ,
 min INTEGER ,
 axis INTEGER ,
 values OCTET STRING }

END
```