Biomolecular Sequence Analysis Specification, V1.0

This OMG document replaces the draft adopted specification and the submission documents. It is an OMG Final Adopted Specification, which has been approved by the OMG board and technical plenaries, and is currently in the finalization phase. Comments on the content of this document are welcomed, and should be directed to *issues@omg.org* by December 4, 2000.

You may view the pending issues for this specification from the OMG revision issues web page *http://www.omg.org/issues/*; however, at the time of this writing there were no pending issues.

The FTF Recommendation and Report for this specification will be published on January 8, 2001. If you are reading this after that date, please download the available specification from the OMG formal specifications web page.

OMG Adopted Specification
November, 2000

Biomolecular Sequence Analysis Specification

Final Adopted Specification
November 2000

Contents

Contents

Contents

Preface

About the Object Management Group

The Object Management Group, Inc. (OMG) is an international organization supported by over 800 members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development. The organization's charter includes the establishment of industry guidelines and object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments. Conformance to these specifications will make it possible to develop a heterogeneous applications environment across all major hardware platforms and operating systems.

OMG's objectives are to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based.

What is CORBA?

The Common Object Request Broker Architecture (CORBA), is the Object Management Group's answer to the need for interoperability among the rapidly proliferating number of hardware and software products available today. Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them. CORBA 1.1 was introduced in 1991 by Object Management Group (OMG) and defined the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction within a specific implementation of an Object Request Broker (ORB). CORBA 2.0, adopted in December of 1994, defines true interoperability by specifying how ORBs from different vendors can interoperate.

Associated OMG Documents

The CORBA documentation is organized as follows:

- *Object Management Architecture Guide* defines the OMG's technical objectives and terminology and describes the conceptual models upon which OMG standards are based. It defines the umbrella architecture for the OMG standards. It also provides information about the policies and procedures of OMG, such as how standards are proposed, evaluated, and accepted.
- *CORBA: Common Object Request Broker Architecture and Specification* contains the architecture and specifications for the Object Request Broker.
- *CORBAservices: Common Object Services Specification* contains specifications for OMG's Object Services.

The OMG collects information for each specification by issuing Requests for Information, Requests for Proposals, and Requests for Comment and, with its membership, evaluating the responses. Specifications are adopted as standards only when representatives of the OMG membership accept them as such by vote. (The policies and procedures of the OMG are described in detail in the *Object Management Architecture Guide*.)

OMG formal documents are available from our web site in PostScript and PDF format. To obtain print-on-demand books in the documentation set or other OMG publications, contact the Object Management Group, Inc. at:

OMG Headquarters
250 First Avenue, Suite 201
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
pubs@omg.org
http://www.omg.org

Acknowledgments

The following companies submitted and/or supported parts of this specification:

- Concept Five Technologies, Inc.
- EMBL-EBI (European Bioinformatics Institute)
- Genome Informatics Corporation
- Millennium Pharmaceuticals, Inc.
- Neomorphic Software, Inc.
- NetGenics, Inc.
- Oxford Molecular Group
- Sanger Centre

Overview 1

Contents

This chapter contains the following sections.

The domain of biomolecular sequence analysis comprises the sub-domains of biological objects and analysis mechanisms. The modules that address these areas are described in the following sections.

1.1 Module DsLSRBioObjects

Biological objects that are central to this specification include **BioSequence**, which is specialized into **NucleotideSequence** and **AminoAcidSequence**. An **Annotation** object is provided, which is specialized into **SeqAnnotation** for usage with **BioSequences**. **SeqAnnotations** can apply to specific parts of a sequence, and the mechanism to refer to these regions is provided by **SeqRegion** and **Interval**. **CompositeSeqRegion** provides the ability to nest **SeqRegions**. **GeneticCode**, associated with an organism, is an auxiliary object needed when translating sequences. The interface **Alignment** and ancillary types are used for representing comparisons between sequences or sequence families. It is also used in describing **SimilaritySearchHits** (i.e., matches found in sequence database, and **Assemblies**). **SearchHit** and **SearchResult** are used primarily for representing the results of similarity searches (e.g., BLAST).

The **Annotation** factory, the **BioSequence** iterators and factories, **SeqAnnotation** iterator, **GeneticCode** factory, and **AlignmentEncoders** and factories are optional interfaces.

The **Annotation** factory, the **BioSequence** iterators and factories, **GeneticCode** factory, and **AlignmentEncoders** and factories are optional interfaces.

1.2 Module DsLSRAnalysis

The **DsLSRAnalysis** module defines the components for supporting sequence analysis through a generic analysis design. The module provides the means to interrogate analyses inputs, output, and functionality. An analysis can be executed asynchronously as well as synchronously based on the client invocation. Executing analyses can be monitored by subscribing to an event channel or polling for state.

1.3 Domain Model

The domain model is expressed in XML. The domain model includes a simple classification of analyses. This is in response to the mandatory requirement of the RFP, and serves to organize the analyses into groups in a way that matches closely with how researchers and bioinformaticists think about and utilize such analyses.

This classification of analyses consists of three broad categories.

- Searching - including similarity searching (e.g., BLAST)
- Alignment - including contig assembly
- Utilities - including molecular weight and GC content

1.4 General Remarks

This document contains a proposal for a standard that addresses the representation of a number of biological objects, as well as mechanisms for analyzing them. A few design principles and patterns that we have used are outlined first.

1.4.1 Objects-by-value

This document makes extensive use of objects-by-value (OBV, OMG Document orbos/98-01-18). This is a new OMG standard for the so-called **valuetype**, which is an entity that is halfway between an IDL **interface** and an IDL **struct**. They are not yet widely supported by all ORBs, but we think they are a very useful construct, as they promise to provide:

- choice: the client can choose to make the object ‘local’ or leave it remote,
- better scalability: only a single round trip transfers the whole state of the object,
- extendibility through inheritance,
- null value semantics.

We have used OBV **valuetypes** essentially as if they were extendible structs, using the following constraints:

- no methods
- all members / attributes are **public**
- inheritance only of **valuetypes** (no “**supports *SomeInterface***”).

Note – Factory methods were not used in the valuetypes. See the appropriate language mapping specifications for details on using ValueFactories.

1.4.2 Returning Multiple Results

If a method has to return a multi-valued result to the caller, there is a design choice of returning these elements directly as a list, or through an iterator, or using a combination of both. We have adopted the latter, hybrid approach to allow the client to choose between the convenience of directly returned lists and the scalability of iterators. The methods having a multi-valued result use have:

- a list return type,
- a parameter **in unsigned long how_many**,
- a parameter **out *AnIterator* the_rest**.

The client specifies that it wishes to receive a list of no more than **how_many** elements as the direct result. The remaining elements, if any, can be retrieved through the iterator returned in the **out** parameter. The iterators allow the retrieval of one element at a time, or several at once. This pattern was in fact taken directly from the **CosPropertyService**, and provides maximum flexibility to client programs.

A multi-valued result, either returned directly or through an iterator, is guaranteed not to contain duplicates. If a multi-valued result type is ordered and iterators are involved, the ordering is the same as that achieved by not having used any iterators.

1.4.3 Identifier

Many entities in molecular biology require ID strings, usually to uniquely identify it in a certain context. The current document also uses strings for ID attributes, but constrains their syntax and semantics to improve interoperability. To make the intended use of these strings clearer,

```
typedef string Identifier;
```

is provided and used in this specification.

1.4.4 Composite Pattern

The **CompositeSeqRegion** valuetype implements the Composite design pattern [Gamma et al., 1995]. This pattern composes entities into tree structures to represent hierarchies. The Composite pattern treats individual objects and composites uniformly.

A biological example using the Composite pattern is a gene being composed of coding regions from a set of exons.

1.4.5 BioObject Immutability

All BioObjects in this specification, with the single exception of **BioSequence**, are immutable. Modifying other BioObjects is considered out of scope for sequence analysis, as defined by the RFP. Since it is clear that the results of many sequence analyses produce information that is frequently attached to sequences as annotations, we do provide the **add_annotation()** method in **BioSequence**.

Implementors are free to choose to support mutable BioObjects, taking responsibility for the associated life cycle issues.

1.4.6 Rationale for Metadata Approach

A number of the initial submissions to the RFP for sequence analysis explored the use of metadata to describe the various types of sequence analyses that might be available to a client. In response to the RFP requirement for the specification of a domain model for sequence analysis, the metadata approaches varied from string descriptors, structs, and arrays of structs to well defined IDL interfaces.

In the process of preparing this RFP response the submitters considered a number of viable approaches to metadata for sequence analyses. There was a strong desire to leverage existing solutions if possible. A predecessor to this specification described the metadata model using valuetype based extensible structs. This approach is carried into the current specification. In recognition of the increasing use of XML to provide data descriptions for application metadata, the specification was enhanced to also support retrieval of XML based metadata. An XML DTD defining the metadata model has been introduced as well. Additionally the mechanism used to fetch metadata descriptions for analyses has been enhanced to support the introduction of new XML based metadata by supplying a tag that identifies the type of metadata described. It is expected that this tag-based retrieval approach could be used to provide access to OCL, XMI, or other formatted metadata in the future. In particular the submitters would have liked to leverage XMI for metadata description but, in the absence of clear examples of its use, chose to adopt a model based on our previous joint submission and to provide for extension in the future.

BSA Modules and Interfaces 2

Contents

This chapter contains the following sections.

2.1 Module DsLSRBioObjects

The analysis of biomolecular sequence information takes place within the broader domain of computational biology. This domain presents a very heterogeneous, rapidly evolving environment that has proven difficult to standardize. To offer a design that is both complete and practical for the field of sequence analysis, this specification includes an IDL specification for **Annotations** and so-called **SeqAnnotations**, which can be likened to Features in the DDBJ/EMBL/GenBank flat file format. These two data components serve to incorporate and organize additional information relevant to the sequence data. Examples include organism source information, biological descriptors, cross-references, molecular characterizations, known sites and variations within the sequence, bibliographic references, and relations to known diseases. **Annotations** and **SeqAnnotations** can also be attached to a sequence to carry new information that is computationally inferred, or experimentally determined. We believe that it is necessary to offer users an easy, extensible interface to organize and link this resulting information to biomolecular sequences either as whole-sequence **Annotations** or region-specific **SeqAnnotations** (Features).

Some of the entities described below use generic types such as **Any** or name-value pairs. It is deemed outside the scope of this document to standardize the types of values that can be contained in these generic types.

Existing standards that can be represented with the current proposal and to some extent have shaped it are: the NCBI datamodels; the DDBJ/EMBL/GenBank Feature Table Document; various sequence file formats (Fasta, EMBL/GenBank, GCG), and various sequence analysis tools (BLAST, FastA, Smith-Waterman, ClustalW, Wise2, Grail, the GCG suite).

The alignment portion of the response is aimed to effectively model all types of **BioSequence** and **BioSequence** related alignment problems in biomolecular sequence analysis. This ranges from the relatively simple cases of a pairwise alignment of two DNA sequences, to the complex case of a profile-HMM compared to genomic DNA.

2.1.1 General

```
//File: DsLSRBioObjects.idl

#ifndef _DS_LSR_BIOOBJECTS_IDL_
#define _DS_LSR_BIOOBJECTS_IDL_

#pragma prefix "omg.org"

#include <CosLifeCycle.idl>
#include <CosPropertyService.idl>

module DsLSRBioObjects
{
        // ...
};

#endif // _DS_LSR_BIOOBJECTS_IDL_
```

#pragma prefix "omg.org"

To prevent name space pollution and name clashing of IDL types, this module (and all modules defined in this specification) uses the pragma prefix that is the OMG's DNS name.

#include <CosLifeCycle.idl>

NucleotideSequence, **AminoAcidSequence**, **Annotation**, **GeneticCode**, **Alignment**, and **SearchResul**t all inherit from **LifeCycleObject**.

#include <CosPropertyService.idl>

Properties are used in **Annotation**, **SearchHit**, and **SearchResult**.

StringList

typedef sequence<string> StringList;	
Description:	Used to pass and return a set of **strings**.

2.1.2 *StrandType*

There is an intrinsic directionality of biological sequence data, which proceeds 5' to 3' for nucleic acids and N-terminal to C-terminal for proteins. For **NucleotideSequences**, **StrandType** provides an indication of whether the **SeqRegion** refers to the original plus-strand, the complementary minus-strand, or both strands of a double-stranded molecule. The **StrandType** values are used in **SeqRegion**.

The following table contains the valid **StrandType** for each type of **BioSequence**.

Table 2-1 Valid StrandTypes

BioSequence Type	**Valid StrandTypes**
BioSequence	**STRAND_NOT_KNOWN**
NucleotideSequence	**STRAND_NOT_KNOWN, STRAND_PLUS, STRAND_MINUS, STRAND_BOTH**
AminoAcidSequence	**STRAND_NOT_APPLICABLE**

The following table contains the matching **StrandTypes** for each type of **StrandType.**

Table 2-2 Matching StrandTypes

StrandType	**Matching StrandTypes**
STRAND_NOT_KNOWN	**STRAND_NOT_KNOWN, STRAND_PLUS, STRAND_MINUS, STRAND_BOTH**
STRAND_NOT_APPLICABLE	**STRAND_NOT_APPLICABLE**
STRAND_PLUS	**STRAND_NOT_KNOWN, STRAND_PLUS, STRAND_BOTH**
STRAND_MINUS	**STRAND_NOT_KNOWN, STRAND_MINUS, STRAND_BOTH**
STRAND_BOTH	**STRAND_NOT_KNOWN, STRAND_PLUS, STRAND_MINUS, STRAND_BOTH**

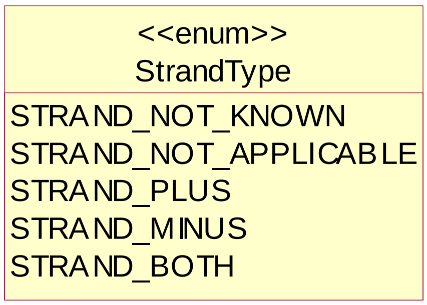


Figure 2-1 The StrandType Enumeration.

enum StrandType {STRAND_NOT_KNOWN, STRAND_NOT_APPLICABLE, STRAND_PLUS, STRAND_MINUS, STRAND_BOTH};

STRAND_NOT_KNOWN	**STRAND_NOT_KNOWN** should be used in all cases not indicated below.
STRAND_NOT_APPLICABLE	**STRAND_NOT_APPLICABLE** should be used for regions of **AminoAcidSequences**.
STRAND_PLUS	**STRAND_PLUS** should be used to indicate the original plus-strand of a **NucleotideSequence**.
STRAND_MINUS	**STRAND_MINUS** should be used to indicate the reverse complement of the plus-strand of a **NucleotideSequence**.
STRAND_BOTH	**STRAND_BOTH** should be used to indicate both strands of a double-stranded **NucleotideSequence**.

2.1.3 Basis

The **Basis** enumeration values are used to specify whether an **Annotation** originated from an experimental result or a computational analysis, such as from the application of a sequence analysis program.

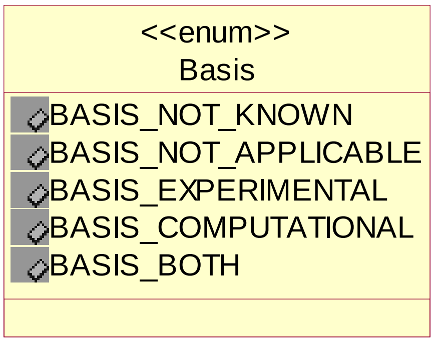


Figure 2-2 The Basis Enumeration

enum Basis {BASIS_NOT_KNOWN, BASIS_NOT_APPLICABLE, BASIS_EXPERIMENTAL, BASIS_COMPUTATIONAL, BASIS_BOTH};

BASIS_NOT_KNOWN	**BASIS_NOT_KNOWN** should be used in all cases not indicated below.
BASIS_NOT_APPLICABLE	**BASIS_NOT_APPLICABLE** should be used to indicate that **Basis** doesn’t apply.
BASIS_EXPERIMENTAL	**BASIS_EXPERIMENTAL** should be used to indicate an experimental result.
BASIS_COMPUTATIONAL	**BASIS_COMPUTATIONAL** is used to indicate a computational analysis, such as from the application of a sequence analysis program.
BASIS_BOTH	Any result determined both experimentally and computationally should use **BASIS_BOTH**.

2.1.4 *Interval*

A contiguous sub-string within a larger string is specified using the **Interval** valuetype. An **Interval** consists of a start and length, defining the starting position of the sub-string and the size of the sub-string (number of units). **BioSequences** are numbered starting at start 1, in keeping with the existing practice in the field of molecular biology. An **Interval** on a **BioSequence** of start=5, length=10 would start at the fifth position and include up to the 14th position of a sequence.

The use of a **start** and **length** instead of start and end provides a powerful mechanism for defining intervals along biological sequences that works well for both linear and circular molecules.

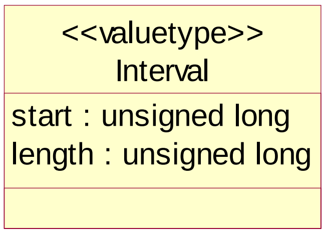


Figure 2-3 The Interval valuetype

```
valuetype Interval
{
      public unsigned long  start;
      public unsigned long  length;
};
```

public unsigned long start;	
Description:	**start** is an unsigned long integer that defines the starting position of the sub-string. **BioSequences** are numbered starting at 1.
Return value:	Returns an **unsigned long**.

public unsigned long length;	
Description:	**length** is an unsigned long integer that defines the size of the sub-string (number of units).
Return value:	Returns an **unsigned long**.

2.1.5 SeqRegion

A **SeqRegion** is a specialization of **Interval** and specifies a location on a **BioSequence**. A further specialization, **CompositeSeqRegion**, may contain zero or more sub-regions. In this specification, **SeqRegion** is used primarily to specify the location along a **BioSequence** to which a **SeqAnnotation** pertains.

The **SeqRegion** model is not intended to address all types of sequence region specification found in the GenBank/EMBL/DDBJ feature table. Supported are intervals with non-fuzzy end points and composites of such intervals. Examples of these include a PROSITE pattern located at 74 and ending at 80, or a gene made of 5 spliced exons.

We believe the definition of **SeqRegion** is broad enough to handle many kinds of commonly occurring sequence-based regions and addresses the needs of most molecular biologists. Due to their complexity and rarity of usage in sequence analysis software, fuzzy sequence regions are not explicitly supported at the present time. It is not currently possible with the present IDL to associate a single **SeqRegion** with a set of **BioSequences**.

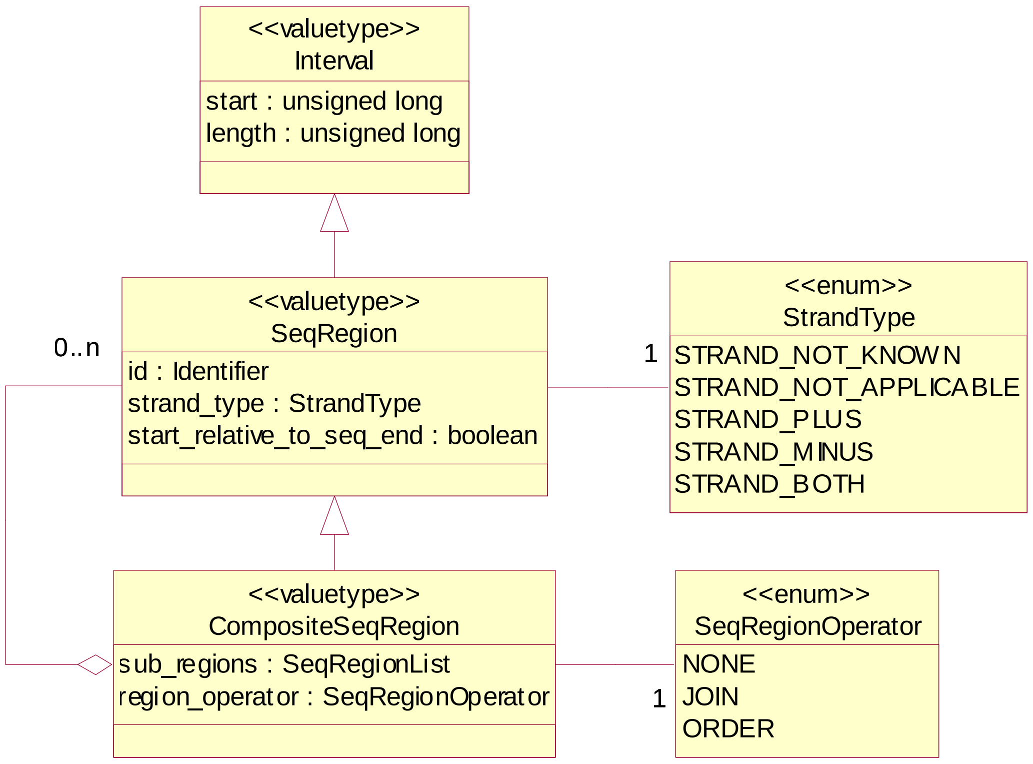


Figure 2-4 The SeqRegion and CompositeSeqRegion Valuetypes

SeqRegion

A **SeqRegion** extends **Interval** and contains the **strand_type** and **start_relative_to_seq_end** members that specialize it for use with biological sequences. **SeqRegion** also contains an **id** indicating the **BioSequence** to which the **SeqRegion** refers.

```
valuetype SeqRegion : Interval
{
    public Identifier  id;
    public StrandType  strand_type;
    public boolean     start_relative_to_seq_end;
};
```

public Identifier id;	
Description:	**id** indicates the **BioSequences** to which the **SeqRegion** refers.
Return value:	Returns an **Identifier**.

public StrandType strand_type;	
Description:	For **NucleotideSequences**, **strand_type** provides an indication of whether the **SeqRegion** refers to the original plus-strand, the complementary minus-strand, or both strands of a double-stranded molecule. **STRAND_MINUS** should be used to indicate a region on the reverse complement of a **NucleotideSequence**. For these regions, **start** and **length** (inherited from **Interval**) refer to positions within the coordinate system of the original, given strand. **strand_type** should be **STRAND_NOT_APPLICABLE** for regions of **AminoAcidSequences**.
Return value:	Returns a **StrandType**.

public boolean start_relative_to_seq_end;	
Description:	The **start_relative_to_seq_end** member can modify the semantics of the start member: if **start_relative_to_seq_end** is TRUE, **start** is to be taken from the end of the sequence, rather than the beginning. No reverse-complement is implied. That is, if sequence has a length 100, and SeqRegion has **start**=20 **length**=10, and **start_relative_to_seq_end** =TRUE, the region runs from position 81 up to and including 90.
Return value:	Returns a **boolean**.

SeqRegionList

typedef sequence<SeqRegion> SeqRegionList;	
Description:	Used to pass a set of **SeqRegions**.

CompositeSeqRegion

CompositeSeqRegion, a specialization of **SeqRegion**, may contain zero or more sub-regions. A **CompositeSeqRegion**'s sub-regions may overlap. The nested or hierarchical behavior is useful in describing complex features on **BioSequences**. There is no limit to nesting. All **CompositeSeqRegions** are expected to be translated in a depth-first traversal, along each node of the tree represented by the **CompositeSeqRegions**. This includes those nodes that have **region_operator** equal to **JOIN** or **ORDER**.

A **CompositeSeqRegion** with sub-regions will itself not have **start**, **length**, **strand_type**, or **start_relative_to_seq_end** data defined. The whole **CompositeSeqRegion** tree will be passed as an object graph by the objects by value (OBV) functionality.

```
valuetype CompositeSeqRegion : SeqRegion
{
    enum SeqRegionOperator
    {
        NONE,       // Region has no sub regions or the sub regions
                    // don't need special treatment.
        JOIN,       // Sub regions should be joined end-to-end to
                    // form a contiguous region.
        ORDER       // Sub region order is important.
    };

    public SeqRegionList      sub_regions;
    public SeqRegionOperator  region_operator;
};
```

enum SeqRegionOperator {NONE, JOIN, ORDER};	
NONE	**NONE** should be used when **JOIN** and **ORDER** are not applicable.
JOIN	**JOIN** should be used when the sub-regions are to be concatenated into a single region.
ORDER	**ORDER** should be used when the sub-regions are to be taken as an ordered set of sub-regions. Typically, it is used to represent a discontinuous region to which a descriptive annotation pertains.

public SeqRegionList sub_regions;	
Description:	**sub_regions** contains the constituent **SeqRegions**. If there are no sub-regions, then **SeqRegion** should be used instead of **CompositeSeqRegion**.
Return value:	Returns a **SeqRegionList**.

public SeqRegionOperator region_operator;	
Description:	The **region_operator** takes on a value of the **SeqRegionOperator** enumeration. It specifies how the sub-regions are to be treated. The sub-regions could be concatenated into a single region (**JOIN**) or taken as an ordered set of sub-regions (**ORDER**). In the latter case, unknown segments of sequence may intervene. All **CompositeSeqRegions** are expected to be translated in a depth-first traversal, along each node of the tree represented by the **CompositeSeqRegions**. This includes those nodes that have **region_operator** equal to **JOIN** or **ORDER**.
Return value:	Returns a **SeqRegionOperator**.

2.1.6 Annotation

The **Annotation** interface defines an annotation that could, in principle, be associated with any bio-object that requires description using name-value pairs.

All attributes in **Annotations** are readonly, in keeping with our immutability policy for this specification.

Annotation inherits from **CosLifeCycle::LifeCycleObject**.

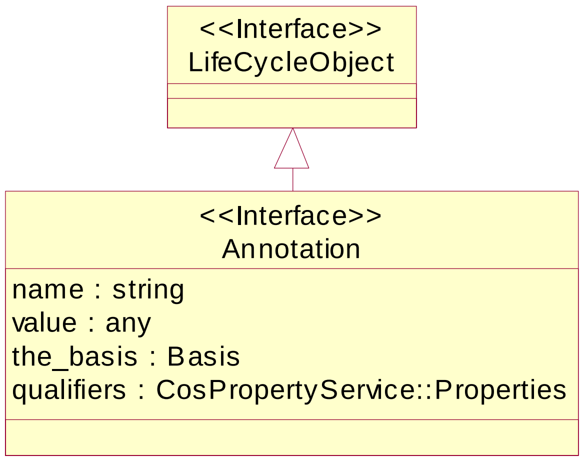


Figure 2-5 The Annotation Interface

```
interface Annotation : CosLifeCycle::LifeCycleObject
{
    readonly attribute string    name;       // type of annotation
    readonly attribute any       value;      // the annotation
    readonly attribute Basis     the_basis;  // basis for annotation
    readonly attribute CosPropertyService::Properties  qualifiers;
};
```

readonly attribute string name;	
Description:	The **name** attribute specifies the general type of the annotation that is contained in the value attribute that contains the annotation itself. The value is of type **any** and therefore could contain anything from a block of free text to a specialized datatype.
Return value:	Returns a **string**. **name** shall not be empty.

readonly attribute any value;	
Description:	The **value** attribute contains the annotation itself.
Return value:	The value is of type **any** and therefore could contain anything from a block of free text to a specialized datatype.

readonly attribute Basis the_basis;	
Description:	**Annotation** has a **basis** attribute, which specifies whether the annotation originated from an experimental result (**BASIS_EXPERIMENTAL**) or a computational analysis (**BASIS_COMPUTATIONAL**), such as from the application of a sequence analysis program. Basis provides for a coarse-grained classification of an Annotation.
Return value:	The value is of type **BASIS**.

readonly attribute CosPropertyService::Properties qualifiers;	
Description:	**Annotation** contains additional information in the form of so-called qualifiers, represented by the **CosPropertyService::Property** struct, which enables them to support many kinds of keyword controlled attributes. These properties are essential for covering the full spectrum of current annotation and feature information.
Return value:	The **qualifiers** attribute is of type **CosPropertyService::Properties** and so provides a place for arbitrary name-value pairs.

AnnotationList

typedef sequence<Annotation> AnnotationList;	
Description:	Used to pass a set of **Annotations**.

IteratorInvalid

```
exception IteratorInvalid
{
    string reason;
};
```

Description:	The **IteratorInvalid** exception is raised for cases where the iterator is no longer valid (e.g., new elements have been added to the underlying collection).
Return value:	Returns a **string** containing the reason that the iterator is invalid.

AnnotationIterator

AnnotationIterator provides a strongly typed iterator for **Annotations**.

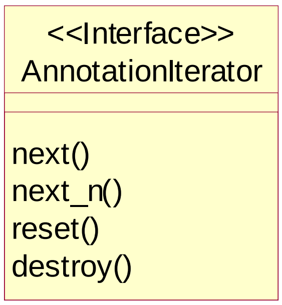


Figure 2-6 The AnnotationIterator Interface

```
interface AnnotationIterator
{
    boolean        next(out Annotation the_annotation)
                       raises(IteratorInvalid);
    boolean        next_n(in unsigned long how_many,
                           out AnnotationList annotations)
                       raises(IteratorInvalid);
    void           reset();
    void           destroy();
};
```

boolean next(out Annotation the_annotation) raises(IteratorInvalid);	
Description:	The **next()** operation gets the next **Annotation** in its out parameter **the_annotation** and returns a boolean value. If the iterator is at the end of the set, it returns FALSE and sets the output **the_annotation** parameter to null.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises **IteratorInvalid** if the iterator is no longer valid (e.g., the underlying collection has changed).

boolean next_n(in unsigned long how_many, out AnnotationList annotations) raises(IteratorInvalid);	
Description:	**next_n()** returns **Annotations** in the **AnnotationList** out parameter **annotations**, containing at most the number specified in the first parameter (**how_many**) and returns a boolean value directly. When it is at the end of the set it returns FALSE and the **annotations** parameter will have length zero. In all cases the length of **annotations** will be the minimum of **how_many** and the number of elements remaining.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises **IteratorInvalid** if the iterator is no longer valid (e.g., the underlying collection has changed).

void reset();	
Description:	**reset()** sets the iterator to the start of the set.
Exceptions:	Raises CORBA::NO_IMPLEMENT with standard minor code 5 if the iterator cannot be reset (e.g., the iterator provides access to streaming data).

void destroy();	
Description:	**destroy()** frees the iterator object.

2.1.7 SeqAnnotation

For biomolecular sequences, **Annotations** are specialized to **SeqAnnotations** to include sequence position information in the form of the **SeqRegion** attribute (see above). Essentially, this attribute indicates to which part of the sequence the annotation pertains, and is analogous to features in the DDBJ/EMBL/GenBank formats. Typical examples include gene, promoter region, and exons.

SeqAnnotation is used to describe an annotation that applies only to a specified region. **Annotation** should be used for an annotation that applies to the associated **BioSequence** as a whole. Although **SeqAnnotations** with null **regions** are also interpreted to apply to the **BioSequence** as a whole, this should be avoided.

SeqAnnotation can associate a BioSequence with analytical results or descriptive information such as biological function. A sequence analysis run could generate **SeqAnnotation** objects as output. In addition, **BioSequence** factories can be used to attach **SeqAnnotations** to the **BioSequences**.

It is not currently possible to navigate from a **SeqAnnotation** to a **BioSequence** using the interfaces defined in this specification. One can, however, obtain a set of **SeqAnnotations** given a **BioSequence**. This is sufficient from the point of view of a sequence analysis application, which could produce annotated sequences. The submitters of this specification feel that there are richer models for annotations on sequences (e.g., complex hierarchies or graphs of relationships between annotations and sequences as well as the annotations themselves). Sequence annotations are expected to be addressed in a future RFP.

To illustrate the uses and coverage of **Annotations** and **SeqAnnotations** with regard to the results of Sequence Analyses, a few more examples are listed below:

- A motif analysis returns a labeled pattern (e.g., KRINGLE) matching a given region of the protein sequence.
- A restriction map analysis returns a list of sites, for the given enzymes, that can then be used to annotate the DNA sequence.
- The result of homology analysis suggests that the sequence belongs to a particular gene family, which can be annotated onto the **NucleotideSequence** including information regarding degree of certainty.
- ORF and gene-finding analyses identify coding regions that are later added as oriented gene features on the sequence.
- Homologous regions found by using an alignment analysis can be annotated as **SeqAnnotations** on the query sequence.
- An EMBL-curated phosphorylation site on a protein stored (imported) as a **SeqAnnotation** on the **AminoAcidSequence**.
- Identified mutations from multiple DNA sequences can be merged into **SeqAnnotations** on a consensus sequence.

Extending **SeqAnnotation** provides a mechanism for creating strongly typed sequence features. This may be appropriate for certain stereotypical sequence features such as genes, exons, and transcriptional regulatory sites that have complex but reasonably well defined semantics. These specialized **SeqAnnotations** could define the necessary data types and sub-feature containment relationships as appropriate for the specific feature.

The issue of annotating **BioSequences** as well as other bio-objects is complex and we are not proposing a definitive solution in the present specification. The proposed IDL is workable for biomolecular sequence analysis and there is sufficient room for elaboration by a future LSR Annotation RFP.

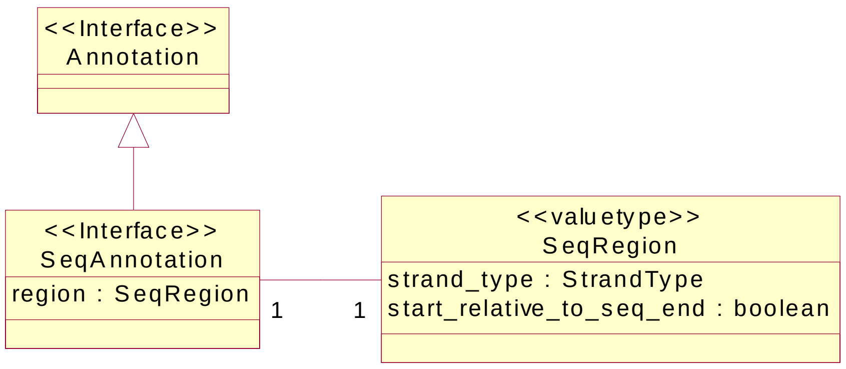


Figure 2-7 The SeqAnnotation Interface

2.1.7.1 *SeqAnnotation Interface*

For biomolecular sequences, **Annotations** are specialized to **SeqAnnotations** to include sequence position information in the form of the **SeqRegion** attribute (see above). If **region** is null, the annotation applies to the associated **BioSequence**(s) as a whole. Otherwise, the annotation applies only to the specified region. **Annotations** should be used instead of **SeqAnnotations** with null **SeqRegions**.

```
interface SeqAnnotation : Annotation
{
    readonly attribute SeqRegion  seq_region;
};
```

readonly attribute SeqRegion seq_region;	
Description:	Contains the sequence position information.
Return value:	Returns a **SeqRegion**.

SeqAnnotationList

typedef sequence<SeqAnnotation> SeqAnnotationList;	
Description:	Used to pass a set of **SeqAnnotations**.

SeqAnnotationIterator

SeqAnnotationIterator provides a strongly typed iterator for **SeqAnnotations**. **SeqAnnotationIterator** is not used directly in this specification, but is provided as a convenience for vendor-specific IDL extensions and future OMG specifications where a collection of **Annotations** contains only **SeqAnnotations**.

SeqAnnotationIterator is an optional interface.

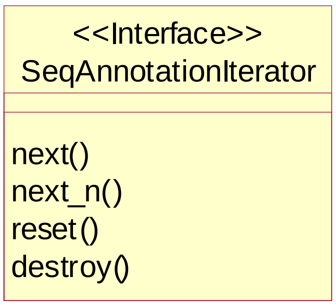


Figure 2-8 The SeqAnnotationIterator Interface

```
interface SeqAnnotationIterator
{
    boolean         next(out SeqAnnotation seq_annotation)
                        raises(IteratorInvalid);
    boolean         next_n(in unsigned long how_many,
                                out SeqAnnotationList seq_annotations)
                        raises(IteratorInvalid);
    void            reset();
    void            destroy();
};
```

boolean next(out SeqAnnotation seq_annotation) raises(IteratorInvalid);	
Description:	The **next()** operation gets the next **SeqAnnotation** in its out parameter **seq_annotation** and returns a boolean value. If the iterator is at the end of the set, it returns FALSE and sets the output **seq_annotation** parameter to null.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises **IteratorInvalid** if the iterator is no longer valid (e.g., the underlying collection has changed).

boolean next_n(in unsigned long how_many, out SeqAnnotationList seq_annotations) raises(IteratorInvalid);	
Description:	**next_n()** returns **SeqAnnotations** in the **SeqAnnotationList** out parameter **seq_annotations**, containing at most the number specified in the first parameter (**how_many**) and returns a boolean value directly. When it is at the end of the set it returns FALSE and the **seq_annotations** parameter will have length zero. In all cases the length of **seq_annotations** will be the minimum of **how_many** and the number of elements remaining.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises **IteratorInvalid** if the iterator is no longer valid (e.g., the underlying collection has changed).

void reset();	
Description:	**reset()** sets the iterator to the start of the set.
Exceptions:	Raises CORBA::NO_IMPLEMENT with standard minor code 5 if the iterator cannot be reset (e.g., the iterator provides access to streaming data).

void destroy();	
Description:	**destroy()** frees the iterator object.

2.1.8 Identifier

There is a need for a data type to indicate an entity's identity in very many situations. In most cases, this need is, or can be addressed by using a string type. The advantages are that it is simple, lightweight, and used universally throughout the realm of computing (and indeed outside). However, the risk of using strings is that they can be too flexible, both in terms of syntax and semantics. This easily results in the lack of interoperability. To allow strings, yet mitigate their potential for abuse, this standard uses the syntax convention of **CosNaming::StringName** as described in the Interoperable Naming service. This convention is mainly a syntactical one; in no way is the use of a naming service implementation required or implied (but it is not precluded either).

2.1.8.1 Identifier Description

A brief description is as follows: **CosNaming::Name** is a list of struct **NameComponents**. (For the purpose of illustration, a **NameComponent** can likened to a directory or filename, whereas **CosNaming::Name** constitutes a full path-name). The struct **NameComponent** has string members id and kind. To transform a **CosNaming::Name** into a string, all its **NameComponents** are represented as strings "*id.kind*."

- If the kind-field is empty, this becomes simply "*id*;"
- if the id-field is empty, this becomes "*.kind*;"
- finally, the Naming service allows both id and kind to be empty, which is represented as ".".

The full stringified **CosNaming::Name** is obtained by concatenating all the **NameComponents** using "/" as a separator character. The character "\" is designated as an escape character; if it precedes any of the special characters ".", "/" and "\", they are taken as literal characters. The typedef string **CosNaming::StringName** is provided for strings used as object names using this convention.

This specification adopts this syntax convention, but requests that the components of the **Identifier** data type adhere to some additional semantic constraints. These rules do not follow from, nor are implied by any semantics of the Naming Service. The additional constraints make this data type sufficiently different from **CosNaming::StringName** to warrant the dedicated typedef string **Identifier**.

typedef string Identifier;	
Description:	In this description, 'component' means: the sub-string of an **Identifier** that corresponds to one **CosNaming::NameComponent**; likewise, id-field and kind-field correspond to the equivalent fields of **NameComponent**.

The rules are as follows:

- Names can refer to collections of entities (such as databases), or to entities within such collections. Names referring to collections consist of exactly one component; names referring to entities within collections consist of at least two components.
- The first component represents the data source. Data sources can be anything: transient collections, local databases, public repositories. It is up to the implementation to document the accepted names for the data source.
- The empty name (".") is valid for the first component, and represents the 'local' or 'default' collection. It is up to the implementation to document what the semantics of 'local' or 'default' is.

- Names that refer to entities within collections consist of two or more components. The second component of such names represents an identifier that is unique in the context of the data source. No empty id-fields are allowed in this or any further components.
- If two components are not enough to uniquely identify an entity, an Identifier can contain more than two components, but no more than necessary to make the identification unique. That is, an Identifier may not be used to freely attach textual information.
- The only characters valid in a component are "a" through "z", "0" through "9", and "-" (hyphen), "_" (underscore), "$" and "." (period). Use of the latter is discouraged since it has a special meaning in the stringifying convention, and has therefore to be escaped.
- To comply with existing practice in the field of public data repositories, it is strongly advised that implementations do string comparisons in a case-insensitive manner. The Naming Service standard fails to mention whether type-case is, for identification purposes, significant or not. Implementations that use a third-party implementation of the Naming Service may therefore wish to restrict Identifiers to only use one type-case. It is up to an implementation to state whether mixed type-case is allowed, and whether type-case is significant in comparisons.

The *id* and *kind* parts of the string components of **Identifier** are used as follows:

- The id-field of a component contains the principal value that makes it unique in the scope provided by the preceding component. It may only be empty in the case of the first component of an Identifier.
- The kind-field of a component is used to represent information indicating the release (for a data source) or version (for an entry) of an entity, and can be empty. If kind is empty and entities with non-empty kind-fields exist, an empty kind field becomes synonymous with the latest release or version. It is up to the implementation to document the syntax and semantics of the version information.

The adoption of this convention has the following advantages:

- it is simple and lightweight,
- it has a well-defined and re-used syntax,
- it is compatible with existing practice,
- it is sufficiently flexible to allow for sub-ids if necessary.

IdentifierList

typedef sequence<Identifier> IdentifierList;	
Description:	Used to pass a set of **Identifiers**.

IdentifierNotFound

```
exception IdentifierNotFound
{
      Identifier id;
};
```

Description:	The **IdentifierNotFound** exception is raised for cases where the database and the identifier within the database can be resolved but the Identifier is not present.
Return value:	Returns the **Identifier** that could not be found.

IdentifierNotResolvable

```
exception IdentifierNotResolvable
{
      Identifier id;
      string reason;
};
```

Description:	The **IdentifierNotResolvable** exception is raised for cases where database and the identifier within the database cannot be resolved such that the **Identifier** cannot even be searched for.
Return value:	Returns the **Identifier** that could not be resolved and a string containing the reason resolution was not possible.

IdentifierNotUnique

```
exception IdentifierNotUnique
{
      Identifier id;
      IdentifierList ids;
};
```

Description:	The **IdentifierNotUnique** exception is raised for cases when the **Identifier** specification is ambiguous and returns more than one object.
Return value:	Returns the non-unique **Identifier** and an **IdentifierList** containing **Identifiers** for all objects that id identifies.

2.1.9 BioSequence

A **BioSequence** is an abstraction of a biological sequence, such as the ordered nucleotides of a DNA chain or the ordered amino acid residues of a protein molecule. A **BioSequence** can be of any length and significance; there is no implication that it corresponds to, for example, a gene. The **BioSequence** interface provides essential characteristics of biological sequences (**name**, **id**, **description**, **length**) and operations for obtaining the sequence string itself or a sub-sequence as an ASCII string of IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) upper case single letter codes (**seq()**, **seq_interval()**).

Additional operations within **BioSequence** provide access to any annotations associated with the **BioSequence** (**get_annotations()**) or the number of annotations (**num_annotations()**).

Annotations can be attached to **BioSequences** directly using the **add_annotation()** method of **BioSequence** or by using the **BioSequence** factories. Thus, **BioSequences** are mutable at the level of their associated annotations. This minimal mutability model permits new annotations to be attached to a **BioSequence** and prevents situations where multiple **BioSequences** might exist on a server with different sets of annotations but representing the same sequence. A NotUpdateable exception can be used to indicate that an **Annotation** cannot be added to this **BioSequence**.

Standard container behavior applies here. If a client destroys a **BioSequence**, it is also up to the client to manage the contents, namely the **Annotations**.

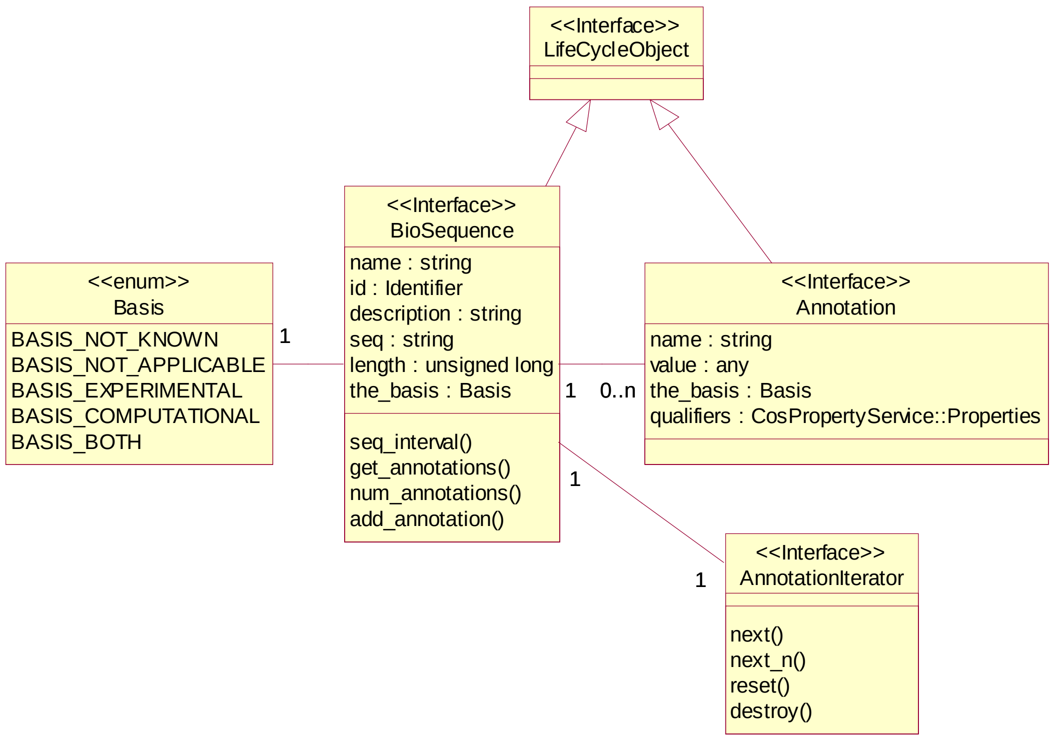


Figure 2-9 The BioSequence Interface

IntervalOutOfBounds

exception IntervalOutOfBounds
{
    Interval invalid;
    Interval valid;
};	

Description:	The IntervalOutOfBounds exception is raised if an Interval's **start** is less than 1 or if its **start**+**length**-1 is greater than the **length** of the **BioSequence**. If a **BioSequence** represents circular DNA, then this exception should be raised if the **Interval's start** is less than 1 or greater than the **length** of the **BioSequence**, or if its **length** is greater than that of the **BioSequence**.
Return value:	Returns the invalid **Interval** and the valid **Interval**. The valid **Interval** has **start** equal to 1 and **length** equal to the length of the **BioSequence**, the largest allowed **Interval**.

SeqRegionOutOfBounds

exception SeqRegionOutOfBounds
{
    SeqRegion invalid;
    Interval valid;
};	

Description:	The SeqRegionOutOfBounds exception is raised if a **SeqRegion**'s **start** is less than 1 or if its start+length-1 is greater than the **length** of the **BioSequence**. The exception is also raised if a nested sub-region of a **CompositeSeqRegion** is invalid. If a **BioSequence** represents circular DNA, then this exception should be raised if the **Interval's start** is less than 1 or greater than the **length** of the **BioSequence**, or if its **length** is greater than that of the **BioSequence**.
Return value:	Returns the invalid **SeqRegion** and the valid **Interval**. The valid **Interval** has **start** equal to 1 and **length** equal to the length of the **BioSequence**, the largest allowed **Interval**.

SeqRegionInvalid

exception SeqRegionInvalid
{
    string reason;
};	

Description:	The SeqRegionInvalid exception is raised if a **SeqRegion** or **CompositeSeqRegion** is invalid in a given context. Examples include the **StrandType** being **STRAND_BOTH** when trying to translate a nucleotide sequence, or a wrong **SeqRegionOperator**, overlaps, or circularity in a **CompositeSeqRegion** passed as an **in** parameter.
Return value:	Returns a string containing the reason the **SeqRegion** is invalid.

NotUpdateable

exception NotUpdateable
{
    string reason;
};	

Description:	The NotUpdateable exception is raised if the **BioSequence** is immutable.
Return value:	Returns a string containing the reason the **BioSequence** cannot be updated.

BioSequence

A **BioSequence** is an abstraction of a biological sequence, such as the ordered nucleotides of a DNA chain or the ordered amino acid residues of a protein molecule. The **BioSequence** interface provides essential characteristics of biological sequences (**name**, **id**, **description**, **length**) and operations for obtaining the sequence string itself or a sub-sequence as an ASCII string of IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) upper case single letter codes (**seq()**, **seq_interval()**).

BioSequence inherits from **CosLifeCycle::LifeCycleObject**.

```
interface BioSequence : CosLifeCycle::LifeCycleObject
{
    readonly attribute string                name;
    readonly attribute Identifier            id;
    readonly attribute string                description;
    readonly attribute string                seq;
    readonly attribute unsigned long         length;
    readonly attribute Basis                 the_basis;

    string              seq_interval(in Interval the_interval)
```

```
                                  raises(IntervalOutOfBounds, SeqRegionInvalid);
      AnnotationList          get_annotations(
                                      in unsigned long how_many,
                                      in SeqRegion seq_region,
                                      out AnnotationIterator the_rest)
                                  raises(SeqRegionOutOfBounds, SeqRegionInvalid);
      unsigned long           num_annotations(in SeqRegion seq_region)
                                  raises(SeqRegionOutOfBounds, SeqRegionInvalid);
      void                    add_annotation(
                                      in Annotation the_annotation)
                                  raises(NotUpdateable, SeqRegionOutOfBounds,
                                      SeqRegionInvalid);
};
```

readonly attribute string name;	
Description:	The **name** attribute represents a human-readable common name for the **BioSequence** (such as a gene name).
Return value:	Returns a **string**.

readonly attribute Identifier id;	
Description:	The **id** attribute represents an ID for the **BioSequence**. Typically a database name and key will be encoded in the **Identifier**.
Return value:	Returns an **Identifier**. **id** shall not be empty

readonly attribute string description;	
Description:	The **description** attribute, a concise description of the **BioSequence**, typically includes functional information, e.g., the contents of the description line from a FASTA file.
Return value:	Returns a **string**.

readonly attribute string seq;	
Description:	The **seq** attribute contains the actual sequence data. The entire sequence is returned. Use **seq_interval()** to access sub-sequences.
Return value:	Returns an ASCII string of IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) upper case single letter codes representing the entire sequence. The string will not contain any termination characters (e.g., '*') or gap characters (e.g., '-').

readonly attribute unsigned long length;	
Description:	The **length** attribute is the length of the **BioSequence**. The **BioSequence** is numbered from 1 to **length**.
Return value:	Returns an **unsigned long**.

readonly attribute Basis the_basis;	
Description:	The **BioSequence basis** attribute can be any of the values of the **Basis** enumeration and specifies whether the sequence has been experimentally determined (**BASIS_EXPERIMENTAL**), computationally determined (**BASIS_COMPUTATIONAL**), or both (**BASIS_BOTH**), or if this information is not known (**BASIS_NOT_KNOWN**). An example of a computational sequence would be a protein sequence that was determined by in silico translation of an experimentally determined DNA sequence.
Return value:	Returns a **Basis** value.

string seq_interval(in Interval the_interval) raises(IntervalOutOfBounds, SeqRegionInvalid);	
Description:	Provides access to sub-sequences of the **BioSequence**. The **Interval** argument indicates which sub-sequence should be returned. The entire sequence may also be obtained using the **seq** attribute. If **the_interval** is a **SeqRegion** and the **StrandType** is **STRAND_MINUS**, the string returned should be taken as reverse-complemented.
Return value:	Returns an ASCII string of IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) upper case single letter codes representing the appropriate sub-sequence.
Exceptions:	Raises IntervalOutOfBounds if the **Interval's start** is less than 1 or if its **start+length**-1 is greater than the length of the **BioSequence**. If the **BioSequence** represents circular DNA, then this exception should be raised if the **Interval's start** is less than 1 or greater than the **length** of the **BioSequence**, or if its **length** is greater than that of the **BioSequence**. Raises SeqRegionInvalid if **the_interval** is a invalid **SeqRegion**. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

AnnotationList get_annotations(in unsigned long how_many, in SeqRegion seq_region, out AnnotationIterator the_rest) raises(SeqRegionOutOfBounds, SeqRegionInvalid);	
Description:	Uses the list/iterator hybrid to provide access to the **Annotations**. A list of no more than **how_many** elements is returned as the direct result. The remaining elements, if any, are available through the iterator returned in the **out** parameter. Only the **SeqAnnotations** that overlap **seq_region** and have compatible **StrandTypes** will be returned. If **seq_region** is null, only **Annotations** are returned.

...continued

Return value:	Returns an **AnnotationList** containing no more than **how_many** elements. The **AnnotationIterator** provides access to any remaining elements.
Exceptions:	Raises SeqRegionOutOfBounds if **seq_region** is out of bounds for this **BioSequence**. Raises SeqRegionInvalid if the **SeqRegion** is invalid. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

unsigned long num_annotations(in SeqRegion seq_region) raises(SeqRegionOutOfBounds, SeqRegionInvalid);	
Description:	Provides access to the number of **Annotations** associated with this **BioSequence**. Only the **SeqAnnotations** that overlap **seq_region** and have compatible **StrandTypes** will be counted. If **seq_region** is null, only **Annotations** are counted.
Return value:	Returns an **unsigned long**.
Exceptions:	Raises SeqRegionOutOfBounds if **seq_region** is out of bounds for this **BioSequence**. Raises SeqRegionInvalid if the **SeqRegion** is invalid. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

void add_annotation(in Annotation the_annotation) raises(NotUpdateable, SeqRegionOutOfBounds, SeqRegionInvalid);	
Description:	**Annotations** can be attached to **BioSequences** directly using the **add_annotation()** method of **BioSequence**.
Exceptions:	Raises NotUpdateable if the **BioSequence** is immutable. Raises SeqRegionOutOfBounds if the **Annotation** is a **SeqAnnotation** and the corresponding **SeqRegion** is out of bounds for this **BioSequence**. Raises SeqRegionInvalid if the **Annotation** is a **SeqAnnotation** that has a **SeqRegion** that is deemed invalid. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

BioSequenceList

typedef sequence<BioSequence> BioSequenceList;	
Description:	Used to pass a set of **BioSequences**.

2.1.10 Sub-types of BioSequence

The data type **BioSequence** is an interface representing biological sequences. All instances of actual biological sequences are expected to derive from one of the **BioSequence** sub-types, **NucleotideSequence** or **AminoAcidSequence** (or specialized sub-types thereof).

Sequence information input to a **BioSequence** or used for querying purposes is case-insensitive. Sequence information output from a **BioSequence** is returned using upper-case ASCII strings of IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) single-letter character codes.

AminoAcidSequence represents a protein sequence and does not contain any operations. A reverse translation operation that produces a nucleic acid sequence from the amino acid sequence is a complex operation that is not straightforward to standardize at this time.

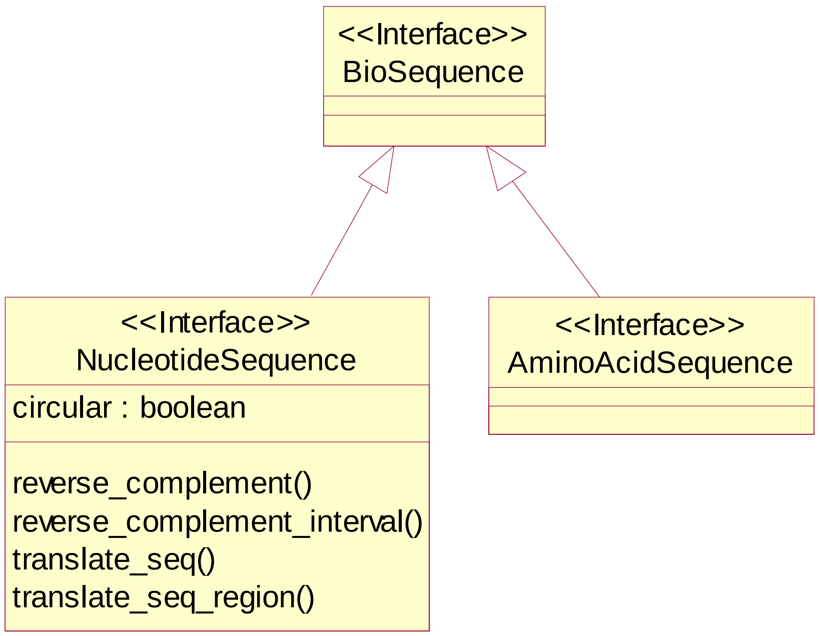


Figure 2-10 The NucleotideSequence and AminoAcidSequence Interfaces

UnsignedLongList

typedef sequence<unsigned long> UnsignedLongList;	
Description:	Used to pass a set of **unsigned longs**.

ReadingFrameInvalid

**exception ReadingFrameInvalid
{
short invalid;
};**	

Description:	The **ReadingFrameInvalid** exception is raised if the reading frame is not between -3 and +3, excluding zero.
Return value:	Returns a **short** containing the invalid reading frame.

NucleotideSequence

NucleotideSequence extends **BioSequence** and represents a DNA or RNA sequence and provides a number of operations for manipulating the sequence data. There is an intrinsic directionality of nucleotide sequence data, from 5' to 3'.

```
interface NucleotideSequence : BioSequence
{
    readonly attribute boolean          circular;
```

```
        string      reverse_complement();
        string      reverse_complement_interval(in Interval the_interval)
                      raises(IntervalOutOfBounds, SeqRegionInvalid);
        string      translate_seq(
                            in short reading_frame,
                            out UnsignedLongList stop_locations)
                      raises(ReadingFrameInvalid);
        string      translate_seq_region(
                            in SeqRegion seq_region,
                            out UnsignedLongList stop_locations)
                      raises(SeqRegionOutOfBounds, SeqRegionInvalid);
};
```

readonly attribute boolean circular;	
Description:	The **circular** attribute provides a mechanism to indicate whether a **NucleotideSequence** is circular, as is the case for plasmids or certain microbial chromosomes.
Return value:	Returns a TRUE if the **NucleotideSequence** is circular and FALSE otherwise.

string reverse_complement();	
Description:	**reverse_complement()** returns an upper-case ASCII string consisting of the reverse complement of the given **NucleotideSequence**.
Return value:	Returns an upper-case ASCII **string**.

string reverse_complement_interval(in Interval the_interval) raises(IntervalOutOfBounds);	
Description:	**reverse_complement_interval()** permits the retrieval of a reverse complement string for a sub-sequence of the given sequence defined by the **Interval** argument. If **the_interval** is a **SeqRegion** and the **StrandType** is **STRAND_MINUS**, the string returned should be taken as reverse-complemented. This will result in a no-op (i.e., the **strand_type** leads to reverse-complementing), which is then reverse-complementing due to the semantics of the method, resulting in the same string that would be returned from **seq_interval**().

...continued

Return value:	Returns an upper-case ASCII **string**.
Exceptions:	Raises IntervalOutOfBounds if the **Interval's start** is less than 1 or its **start+length-1** is greater than the **length** of the **NucleotideSequence**. If the **NucleotideSequence** represents circular DNA, then this exception should be raised if the **Interval's start** is less than 1 or greater than the **length** of the **NucleotideSequence**, or if its **length** is greater than that of the **NucleotideSequence**. Raises SeqRegionInvalid if **the_interval** is an invalid **SeqRegion**. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

```
string translate_seq(
                    in short reading_frame,
                    out UnsignedLongList stop_locations)
    raises(ReadingFrameInvalid);
```

Description:	**translate_seq()** returns a string representing the conceptual amino acid translation of the nucleic acid sequence. **translate_seq()** requires the reading frame in which the translation is to be performed. The **reading_frame** should be a signed integer (**short**) between -3 and +3, excluding zero. If **reading_frame** is positive, (reading_frame - 1) nucleotides at the beginning (5' end) of the sequence are ignored. If **reading_frame** is negative, its absolute value should be applied to the 5' end of the complementary (minus) strand.
Return value:	The returned **string** consists of upper-case single-letter IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) character codes for the translated amino acids. Any internal stop codons are represented by '*'. The **UnsignedLongList out** parameter **stop_locations** contains the locations of any internal stops (terminators) in the protein translation.
Exceptions:	Raises ReadingFrameInvalid if **reading_frame** is not between -3 and +3, excluding zero.

string translate_seq_region(in SeqRegion seq_region, out UnsignedLongList stop_locations) raises(SeqRegionOutOfBounds, SeqRegionInvalid);	
Description:	**translate_seq_region()** performs a translation of a defined region of a **NucleotideSequence** specified by the **SeqRegion** argument. No reading frame is necessary because the **SeqRegion** defines the frame. A **SeqRegion** is required here instead of an interval because non-contiguous segments of a **NucleotideSequence** may need to be specified, as in the case of a DNA sequence containing introns. If a region submitted for translation contains sub-regions, all sub-regions are concatenated in depth-first order prior to translation. If **seq-region's StrandType** is **STRAND_MINUS**, the string returned should be taken as reverse-complemented before being translated.
Return value:	The returned **string** consists of upper-case single-letter IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) character codes for the translated amino acids. Any internal stop codons are represented by '*'. The **UnsignedLongList out** parameter **stop_locations** contains the locations of any internal stops (terminators) in the protein translation.
Exceptions:	Raises SeqRegionOutOfBounds if any contained **Interval's start** is less than 1 or its **start+length-1** is greater than the length of the **NucleotideSequence**. If the **NucleotideSequence** represents circular DNA, then this exception should be raised if the **SeqRegion's start** is less than 1 or greater than the **length** of the **NucleotideSequence**, or if its **length** is greater than that of the **NucleotideSequence**. Raises SeqRegionInvalid if **seq_region** is invalid. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

NucleotideSequenceList

typedef sequence<NucleotideSequence> NucleotideSequenceList;	
Description:	Used to pass a set of **NucleotideSequences**.

AminoAcidSequence

AminoAcidSequence extends **BioSequence** and represents a protein sequence and does not contain any operations. A reverse translation operation that produces a nucleic acid sequence from the amino acid sequence is a complex operation that is not straightforward to standardize at this time. There is an intrinsic directionality of protein sequence data, from N-terminal to C-terminal.

```
interface AminoAcidSequence : BioSequence
{
};
```

AminoAcidSequenceList

typedef sequence<AminoAcidSequence> AminoAcidSequenceList;	
Description:	Used to pass a set of **AminoAcidSequences**.

2.1.11 CodeRule

CodeRule is a valuetype that defines the correspondence between a **Codon** and a **Residue** type. The **Residue** member (residue) is a single ASCII character representing an amino acid in the IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) standard. The **Codon** member (codon) is an array of three **Bases**, which are characters representing unambiguous nucleotides using the IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) symbols for nucleotide nomenclature (see References).

Residue

typedef char Residue;	
Description:	The **Residue** member (residue) is a single ASCII character representing an amino acid using the IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) symbols for amino acid nomenclature (see References).

Base

typedef char Base;	
Description:	A **Base** is a character representing an unambiguous nucleotide using the IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) symbols for nucleotide nomenclature (see References).

Codon

typedef Base Codon[3];	
Description:	A **Codon** is an array of three **Bases**.

CodeRule

CodeRule is a valuetype that defines the correspondence between a **Codon** and a **Residue** type.

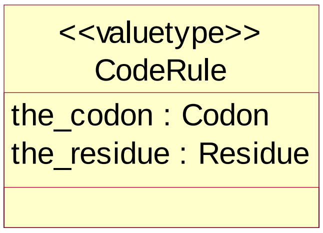


Figure 2-11 The CodeRule valuetype

```
valuetype CodeRule
{
	public Codon	the_codon;
	public Residue	the_residue;
};
```

public Codon the_codon;	
Description:	The **Codon** member (codon) is an array of three **Bases**, which are characters representing unambiguous nucleotides using the IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) symbols for nucleotide nomenclature (see References).
Return value:	Returns a **Codon**.

public Residue the_residue;	
Description:	The **Residue** member (residue) is a single ASCII character representing an amino acid using the IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) symbols for amino acid nomenclature (see References).
Return value:	Returns a **Residue**.

Coding

typedef CodeRule Coding[64];	
Description:	A **Coding** is an array of sixty-four **CodeRules**. Sixty-four is the number of combinations of the four **Bases** (A, G, C, U) taken three at a time.

GeneticCodeName

typedef string GeneticCodeName;	
Description:	A **GeneticCodeName** is a **string** that contains the name of a currently known genetic code.

GeneticCodeNameList

typedef sequence<GeneticCodeName> GeneticCodeNameList;	
Description:	Used to pass a set of **GeneticCodeNames**.

InvalidResidue

```
exception InvalidResidue
{
    Residue the_residue;
    unsigned long offset;
};
```

Description:	The InvalidResidue exception is raised if the **Residue** is inconsistent with the IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) single letter codes. Note that residue may be interpreted to mean base (see Glossary).
Return value:	Returns the invalid **Residue** and its offset within the **BioSequence**.

2.1.12 GeneticCode

The **GeneticCodeFactory** interface defines a set of **const GeneticCodeName** strings that list the set of currently known genetic codes. A **GeneticCode** object should be created with its name member set to one of these **GeneticCodeNames**. The **GeneticCode** object is used for translating a string of nucleic acid bases into a string of amino acid residues. The **GeneticCodeName** defines the particular Coding that is used to convert **Codons** into **Residues** so one need only specify the **GeneticCodeName** when creating a **GeneticCode** object from one of the known types. Codings for the **GeneticCodeNames** listed below in **GeneticCodeFactory** can be found in Appendix B "Genetic Codes".

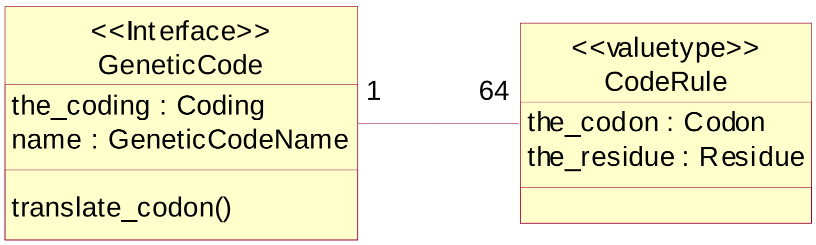


Figure 2-12 The GeneticCode interface

```
interface GeneticCode : CosLifeCycle::LifeCycleObject
{
    readonly attribute Coding           the_coding;
    readonly attribute GeneticCodeName name;

    Residue translate_codon(in Codon the_codon)
        raises(InvalidResidue);
};
```

readonly attribute Coding the_coding;	
Description:	The **coding** attribute consists of an array of 64 **CodeRules**, which allows the **GeneticCode** object to be used for translating a string of nucleic acid bases into a string of amino acid residues. **Codings** for the **GeneticCodeNames** listed below in **GeneticCodeFactory** can be found in Appendix B.
Return value:	Returns a **Coding**.

readonly attribute GeneticCodeName name;	
Description:	The **name** attribute should be one of the known **GeneticCodeNames** listed in **GeneticCodeFactory**. If the desired genetic code is not represented, an appropriate name should be used.
Return value:	Returns a **GeneticCodeName**.

Residue translate_codon(in Codon the_codon) raises(InvalidResidue);	
Description:	**translate_codon**() uses **coding's** array of sixty-four **CodeRules** to translate a string of nucleic acid bases into a string of amino acid residues.
Return value:	Returns a **Residue**.
Exceptions:	Raises InvalidResidue if the **codon** is inconsistent with the IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) single letter codes. Note that residue is interpreted to mean base here (see Glossary).

2.1.13 AlignmentElement

An **AlignmentElement** corresponds to one 'row' in a traditional alignment. However to make it general, it is represented by a wrapper that allows any **Object** to be used in an **Alignment**. This approach allows the occurrence of one and the same **Object** in different 'rows' (using the **key**), and also avoids the combinatorial problem of having every type of **BioSequence** duplicated just so it can be used in an **Alignment**. This approach allows other objects, not yet defined in this standard (e.g., hidden Markov models, to be used in the alignment). Most commonly, however, **AlignmentElement** will contain an **element** of type **BioSequence**.

The **key** provides a unique reference to each **AlignmentElement** to be maintained between the client and the server of the **Alignment**. Notice that there may be more than one copy of a particular **Object** in the **Alignment**. There is no proscribed semantics to how the **key** is structured. The following provides examples of **keys** that could be used if the **Objects** are **BioSequences**.

Table 2-3 Key Examples

Unique BioSequence Identifiers		
Identifiers	**Example Key Set 1**	**Example Key Set 2**
emb/X04427	emb/X04427	0
emb/XX1111	emb/XX1111	1

Table 2-3 Key Examples

emb/X75541	emb/X75541	2
emb/Y10276	emb/Y10276	3
emb/X95248	emb/X95248	4
Non-unique BioSequence Identifiers (repeated sequence)		
Identifiers	**Example Key Set 1**	**Example Key Set 2**
emb/X04427	emb/X04427	0
emb/XX1111	emb/XX1111	1
emb/X75541	emb/X75541	2
emb/Y10276	emb/Y10276	3
emb/X95248	emb/X95248/0	4
emb/X95248	emb/X95248/1	5

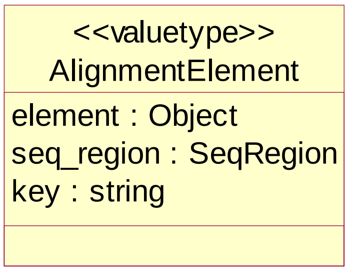


Figure 2-13 The AlignmentElement valuetype.

AlignmentElement

```
valuetype AlignmentElement
{
    public Object       element;
    public SeqRegion    seq_region;
    public string       key;
};
```

public Object element;	
Description:	The analysis that constructs the **Alignment** is responsible for determining if the **Object** is appropriate in the given context. Most commonly, **AlignmentElement** will simply contain an **element** of type **BioSequence**.
Return value:	Returns an **Object**.

public SeqRegion seq_region;	
Description:	The **seq_region** represents the coordinates of a particular segment of the **element** (typically a **BioSequence**) that is aligned in the current **Alignment**, and that is considered one ‘row’ in the **Alignment**. The coordinates are those of the original **Object**, not those of the **Alignment**. Notice that a particular **Object** might be represented more than once in the **Alignment**, and **seq_region** will provide the information as to the region of the **Object** that is used. The only valid **SeqRegionOperator** is **JOIN**.
Return value:	Returns a **SeqRegion**.

public string key;	
Description:	The key provides a unique reference to each **AlignmentElement** to be maintained between the client and the server of the **Alignment**. Notice that there may be more than one copy of a particular **Object** in the **Alignment**. There is no proscribed semantics to how the **key** is structured. It is used in the **get_seq_region()** method in **Alignment** to provide a unique key for this **AlignmentElement**.
Return value:	Returns a **string**.

AlignmentElementList

typedef sequence<AlignmentElement> AlignmentElementList;	
Description:	Used to pass a set of **AlignmentElements**.

2.1.14 AlignmentElementIterator

AlignmentElementIterator provides a strongly typed iterator for **AlignmentElements**.

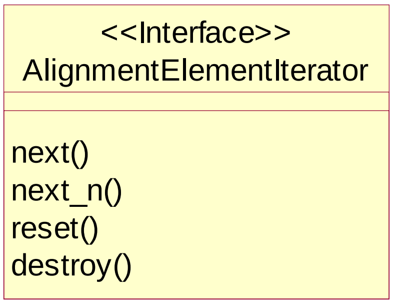


Figure 2-14 The AlignmentElementIterator Interface.

```
interface AlignmentElementIterator
{
    boolean        next(out AlignmentElement element)
                       raises(IteratorInvalid);
    boolean        next_n(in unsigned long how_many,
                           out AlignmentElementList elements)
                       raises(IteratorInvalid);
    void           reset();
    void           destroy();
};
```

boolean next(out AlignmentElement element) raises(IteratorInvalid);	
Description:	The **next()** operation gets the next **AlignmentElement** in its out parameter **element** and returns a boolean value. If the iterator is at the end of the set, it returns FALSE and sets the output **element** parameter to null.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises IteratorInvalid if the iterator is no longer valid (e.g., the underlying collection has changed).

boolean next_n(in unsigned long how_many, out AlignmentElementList elements) raises(IteratorInvalid);	
Description:	**next_n()** returns **AlignmentElements** in the **AlignmentElementList out** parameter **elements**, containing at most the number specified in the first parameter (**how_many**) and returns a boolean value directly. When it is at the end of the set it returns FALSE and the **elements** parameter will have length zero. In all cases the length of **elements** will be the minimum of **how_many** and the number of elements remaining.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises IteratorInvalid if the iterator is no longer valid (e.g., the underlying collection has changed).

void reset();	
Description:	**reset()** sets the iterator to the start of the set.
Exceptions:	Raises CORBA::NO_IMPLEMENT with standard minor code 5 if the iterator cannot be reset (e.g., the iterator provides access to streaming data).

void destroy();	
Description:	**destroy()** frees the iterator object.

2.1.15 Alignment

An **Alignment** is built from a set of correspondences of regions of sequences. In many cases the sequence region is only a single residue (a single base or a single amino acid) long, but this need not be. For example, a region of three DNA base pairs, representing a single amino acid, is a common region size. Each correspondence, which is called a ‘column’ due to the common visual interpretation of an alignment, indicates that a particular region of one sequence is in some manner equivalent to set of particular regions on other sequences. The exact nature of this equivalence differs between different alignment methods, the most common being that these regions shared a common evolutionary ancestor. An alternative is that these regions were read from the same region of physical DNA, as in a DNA assembly.

Alignment representation in sequence analysis has been dominated by text based representation of the correspondences as columns, with sequences running horizontally and each correspondence being represented by a column. Padding characters (often '-') are placed in sequences to align the residues with the correct correspondences in other sequences.

Table 2-4 Multiple Alignment of AminoAcidSequences

seq1	10	RSDGFAFVEF	19
seq2	15	RT-GFAYVEM	23
seq3	20	RTHGFAFVEM	29

```
Correspondence  1: (Seq1, position 10, Seq2, position 15, Seq3 position 20)
Correspondence  2: (Seq1, position 11, Seq2, position 16, Seq3 position 21)
Correspondence  3: (Seq1, position 12, Seq2, none,        Seq3 position 22)
...
Correspondence 10: (Seq1, position 19, Seq2, position 23, Seq3 position 29)
```

This provides a compact representation of the alignment, but relies heavily on single characters being the basis of the correspondence, which makes representing more complex but still common types of alignment challenging. Examples include alignments of DNA and protein sequences and alignments of profile Hidden Markov Models and protein sequences. In addition, text based representation cannot convey any additional information about the nature of the correspondence, which is an issue for more complex alignments. A final drawback to this method of representing an alignment is that it is generally hard to examine only part of the alignment, as the entire text must be processed before the correspondences between positions can be represented explicitly in computer terms.

An IDL definition of an alignment can provide a much richer description of an alignment, but it must be kept in mind that the most common use of an alignment will be to view it, probably in a form very close to Table 2-4. Generating a similar text representation must be simple operation for a client of the **Alignment** interface.

For complex alignments it is convenient to associate with each correspondence the assumption on which the correspondence is made. For example, when aligning a protein sequence to a DNA sequence, it is important to be able to distinguish insertions in the DNA sequence which are due to sequencing errors in the determination of the DNA sequence and insertions due to the evolutionary insertion of bases in the DNA sequence. This implies that each correspondence needs an indication of the assumptions made for the grouping of regions on sequences. Such assumptions are generally made during the alignment process. As such, they are not a fixed property of one particular sequence in the alignment, but they rather belong to the alignment as a whole. Therefore, it is better to associate the assumption(s) with the correspondences, rather than with the sequences.

Although many of the alignments involve **BioSequences**, there are a number which also involve other objects, such as regular expressions and hidden Markov models. These objects are not part of this specification, and, in any case, it is unlikely that any specification could cover all possible objects that will be designed in this field. The proposed specification can handle any CORBA object through the **AlignmentElement** wrapper.

The proposed **Alignment** interface can model simple and complex alignments in a complete way. The object provides accessors to retrieve all the correspondences and the individual regions inside a correspondence. There is no explicit **correspondence** or **column** object, as it seems of little value. Users will generally be using a set of correspondences (i.e., an alignment).

We recognize that there are many uses of an alignment where the client does not want to process the actual alignment information itself, but simply wants to display it to a user or pass it onto programs which are based around old text based alignment formats. The optional **CharacterAlignmentEncoder** interface provides a way for a client to get a more traditional view of an **Alignment**. In addition, this interface lets the server take responsibility for the representation of an **Alignment**. This way, servers can offer clients a complete solution, including representation. For complex alignments that are non-trivial to render, this is an important mechanism. We cannot stress too highly that the representation of an **Alignment**, especially that of gaps, is the job of the **CharacterAlignmentEncoder** and not that of the corresponding **Alignment**.

AlignmentObjectInvalid

```
exception AlignmentObjectInvalid
{
    Object element;
    string reason;
};
```

Description:	The AlignmentObjectInvalid exception is raised if the **Object** is not valid for this **Alignment**. This exception will be raised by analyses that construct **Alignments**.
Return value:	Returns the invalid **Object** and a **string** containing the reason the element is invalid.

ElementNotInAlignment

```
exception ElementNotInAlignment
{
};
```

Description:	The ElementNotInAlignment exception is raised if the **AlignmentElement** is not associated with this **Alignment**..

IndexOutOfBounds

exception IndexOutOfBounds
{
unsigned long invalid;
Interval valid;
};	

Description:	The IndexOutOfBounds exception is raised if an index is out of bounds.
Return value:	Returns the invalid **unsigned long** and the valid **Interval**. The valid **Interval** contains the largest allowed **Interval** for the index.

Alignment

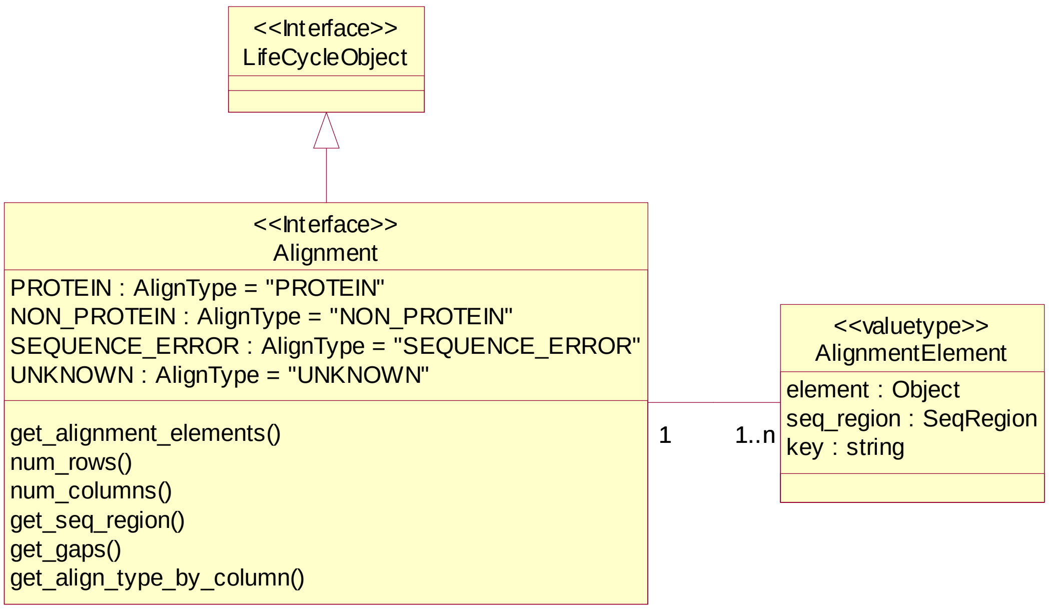


Figure 2-15 The Alignment Interface

```
interface Alignment : CosLifeCycle::LifeCycleObject
{
	typedef string AlignType;
	typedef sequence<AlignType> AlignTypeList;

	const AlignType PROTEIN            = "PROTEIN";
	const AlignType NON_PROTEIN        = "NON_PROTEIN";
	const AlignType SEQUENCE_ERROR = "SEQUENCE_ERROR";
	const AlignType UNKNOWN            = "UNKNOWN";

	AlignmentElementList get_alignment_elements(
                        in unsigned long start,
                        in unsigned long how_many,
                        out AlignmentElementIterator the_rest)
        raises(IndexOutOfBounds);

        unsigned long num_rows();
        unsigned long num_columns();

        SeqRegion get_seq_region(
                        in string key,
                        in Interval the_interval)
                raises(ElementNotInAlignment, IntervalOutOfBounds,
                        SeqRegionInvalid);

        CompositeSeqRegion get_seq_region(
                        in string key,
                        in Interval the_interval)
                raises(ElementNotInAlignment, IntervalOutOfBounds,
                        SeqRegionInvalid);

        AlignType get_align_type_by_column(in unsigned long col)
                raises(IndexOutOfBounds);
};
```

typedef string AlignType;	
Description:	An **AlignType** is a **string** that contains the type of the assumption made for this grouping of regions on sequences. Several kinds of **AlignTypes** are given below.

typedef sequence<AlignType> AlignTypeList;	
Description:	Used to pass a set of **AlignTypes**.

```
const AlignType PROTEIN          = "PROTEIN";
const AlignType NON_PROTEIN      = "NON_PROTEIN";
const AlignType SEQUENCE_ERROR = "SEQUENCE_ERROR";
const AlignType UNKNOWN          = "UNKNOWN";
```

Description:	Common alignment assumptions are provided as simple strings, with constant types as a starting point for a list of assumptions. **UNKNOWN** indicates that no additional information is provided with the alignment, as would be the case for (e.g., Smith-Waterman alignments). **PROTEIN** indicates that this column does encode (part of) a protein. This can be either because it contains one or more amino acid residues, or more importantly, because the column consists of triplet(s) of DNA bases that encode amino acid(s). A very common region size is 1 for amino acids, and 3 for nucleotide triplets. However, more complex regions, e.g., a transmembrane protein sequence segment, are entirely possible. **SEQUENCE_ERROR** indicates that the column contains bases that are considered to be erroneous. For example, in aligning a protein to a DNA sequence it possible to distinguish insertions due to evolutionary processes (**PROTEIN)** from insertions due to sequencing error (**SEQUENCE_ERROR**). More involved alignment methods, for example hidden Markov models, could use the **AlignType** string to provide a sensible decoding of the alignment, and in these cases, the **AlignType** maybe more informative than the **SeqRegion** provided by the **Alignment**.

AlignmentElementList get_alignment_elements(in unsigned long start, in unsigned long how_many, out AlignmentElementIterator the_rest) raises(IndexOutOfBounds);	
Description:	This method allows the retrieval of **AlignmentElements**. They correspond to the rows in a traditional textually represented alignment; typically, the **AlignmentElements** are sequences. Uses the list/iterator hybrid to provide access to the **AlignmentElements**. A list of no more than **how_many** elements starting at **start** is returned as the direct result. The remaining elements, if any, are available through the iterator returned in the **out** parameter. This is particularly useful for **Assemblies**, where for a particular region, only a few sequences from thousands are relevant.
Return value:	Returns an **AlignmentElementList** containing no more than **how_many** elements starting at **start**. The **AlignmentElementIterator** provides access to any remaining elements to the right of those in **AlignmentElementList**.
Exceptions:	Raises IndexOutOfBounds if **start** is less than 1 or more than the number of aligned elements. This upper limit is returned by **num_rows**().

unsigned long num_rows();	
Description:	The **Alignment** interface provides access to the **AlignmentElements** that make up the alignment. The **key** data member uniquely identifies an **AlignmentElement** within the **Alignment**. The total number of **AlignmentElements** is given by **num_rows**().
Return value:	Returns an **unsigned long**.

unsigned long num_columns();	
Description:	The **Alignment** interface provides access to the correspondences that make the alignment. The correspondences are numbered 1 to **length** inclusive, and can be considered the equivalent of alignment columns in a traditional text view of an alignment. The total number of correspondences is given by **num_columns**()
Return value:	Returns an **unsigned long**.

SeqRegion get_seq_region(in string key, in Interval the_interval) raises(ElementNotInAlignment, IntervalOutOfBounds, SeqRegionInvalid);	
Description:	The input parameter **key** unambiguously identifies an **AlignmentElement** within the **Alignment**. For each correspondence, each **AlignmentElement** will have a particular **SeqRegion**, returned by **get_seq_region()**. A null **SeqRegion** indicates that there is no region for this correspondence (i.e., a gap). Multiple gaps are represented by multiple **SeqRegions**. To find the "length" of a gap, it is necessary to check other correspondences in the column. A null **SeqRegion** contains no length information. The input parameter **the_interval** represents an interval in the coordinates of the **Alignment**, not that of the underlying **Object**. If the interval includes a gap at the start, middle or end, the returned **SeqRegion** does not show it, because the **start** and **end** of it are in the coordinate system of the underlying **Object** which is unaware of any gaps. Instead, the corresponding segment of the underlying **Object** is indicated. It is assumed that the numbering of the correspondences is relevant, i.e., that the second correspondence comes after the first, with all the intervals abutting. This allows an **Interval** of correspondences to be a valid concept.
Return value:	Returns a **SeqRegion**.
Exceptions:	Raises ElementNotInAlignment if the **AlignmentElement** is not associated with this **Alignment**. Raises IntervalOutOfBounds if the **Interval's start** is less than 1 or if its **start+length-1** is greater than the total number of correspondences given by **num_columns**(). Raises SeqRegionInvalid if **the_interval** is an invalid **SeqRegion**. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

```
CompositeSeqRegion get_gaps(
                        in string key,
                        in Interval the_interval)
      raises(ElementNotInAlignment, IntervalOutOfBounds,
            SeqRegionInvalid);
```

Description:	The input parameter **key** unambiguously identifies an **AlignmentElement** within the **Alignment**. For each correspondence, each **AlignmentElement** will have gaps returned by **get_gaps()**. A null **CompositeSeqRegion** indicates that there are no gaps. The input parameter **the_interval** represents an interval in the coordinates of the **Alignment**, not that of the underlying **Object**. It is assumed that the numbering of the correspondences is relevant (i.e., that the second correspondence comes after the first, with all the intervals abutting). This allows an **Interval** of correspondences to be a valid concept.
Return value:	Returns a **CompositeSeqRegion**. The **start** and **length** of each constituent **SeqRegion** indicates the location, in the coordinate system of the underlying **Object**, and number of gaps at that location. Gaps of length 0 are not allowed. A start equal to 0 would be before the first base. A start equal to N is a gap between bases N and N+1. So start equal to the **Object's** length would be after the last base.
Exceptions:	Raises ElementNotInAlignment if the **AlignmentElement** is not associated with this **Alignment**. Raises IntervalOutOfBounds if the **Interval's start** is less than 1 or if its **start+length-1** is greater than the total number of correspondences given by **num_columns**(). Raises SeqRegionInvalid if **the_interval** is an invalid **SeqRegion**. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

AlignType get_align_type_by_column(in unsigned long col) raises(IndexOutOfBounds);	
Description:	**get_align_type_by_column**() provides a mechanism to retrieve the assumptions used for this correspondence from the **Alignment**. There is not additional machinery in an **Alignment** itself to help interpret these **AlignTypes**. For specific instances of an **Alignment** constructor, a client that uses the constructor should read the documentation as to how to interpret the **AlignType**, as it will be part of definition of what the **Alignment** constructor actually provides. For clients that do not want to interpret the **Alignment** but would like a sensible representation of it to pass onto other programs or visually to a user, the **AlignmentEncoders**, **CharacterAlignmentEncoder** and **SingleCharacterAlignmentEncoder** will provide an entire server-side solution for the client.
Return value:	Returns an **AlignType**.
Exceptions:	Raises IndexOutOfBounds if **col** is less than 1 or greater than the total number of correspondences given by **num_columns**().

AlignmentList

typedef sequence<Alignment> AlignmentList;	
Description:	Used to pass a set of **Alignments**.

2.1.16 Alignment Examples

The precise interpretation of this specification for alignments is illustrated with a number of examples. Firstly a standard protein multiple alignment is provided, and secondly a more complicated, protein to EST sequence tag alignment is presented.

Protein Multiple Alignment

This alignment is a fragment of an alignment from the Pfam database. A text representation of this alignment is given below.

```
CAJ1_YEAST/6-24    EYYDILGIKP-------EATPTEIKK
YIS4_YEAST/6-24    EYYDLLGVST-------TASSIEIKK
YNW7_YEAST/4-22    CYYELLGVET-------HASDLELKK
YGM8_YEAST/79-104  NLYDVLELPTPLDVHTIYDDLPQIKR
```

The Alignment object which represented this would return four **AlignmentElement** objects from the **get_alignment_elements**() method. The first object would have the **AminoAcidSequence** Object that presented the sequence CAJ1_YEAST in the element attribute and the **SeqRegion** would have the **start** attribute of 6 and a **length** attribute of 19. Calling the **get_seq_region**() method with this **AlignmentElement** and an **Interval** of **start** 1, **length** 1 would provide a **SeqRegion** with **start** 6, **length** 1, being the sixth residue in CAJ1_YEAST, a Glutamate 'E'. The following table shows the results of this call to **get_seq_region**() and several other similar calls, each with different input **Intervals**. All calls are for the sequence CAJ1_YEAST.

Table 2-5 Call Results

input **Interval**		output **SeqRegion**		**string**
start	**length**	**start**	**length**	
1	1	6	1	a Glutamate 'E’
2	1	7	1	a Tyrosine 'Y’
2	3	7	3	the peptide "YYD"
11	1	null		a gap '-'
12	1	null		a gap '-'
12	10	16	4	the peptide "EATP"

The **get_align_type_by_column()** method would return either **UNKNOWN** or **PROTEIN** depending on the implementor. Potentially, if the alignment had been made with a more involved method, for example, a hidden Markov model with a notion of structural state, the structural state that was used in each column could be returned.

Of course, for clients whose main purpose is display, the laborious business of querying each position for the region and then looking into the sequence object for the residue at that position is a convoluted route for retrieving the information. If the implementor provided a **CharacterAlignmentEncoder** for this **Alignment**, then a text representation of the **Alignment** could be quickly retrieved and displayed, potentially using the large-scale transport methods provided in **SingleCharacterAlignmentEncoder** as this alignment has a single character per correspondence. Once displayed, a client could quickly interpret a query on a particular character in the alignment, as it would simply have to call **get_seq_region()** with the column position to retrieve the position in the sequence.

Protein vs. EST alignment

This example is of a drosophila protein compared to an EST sequence with a frame-shift error occurring, as one would find in GCG's FrameSearch, FASTX, and Wise2. A fragment of the alignments is shown in the following table.

Table 2-6 Protein vs. EST Alignment

column	20	21	22	23
EST	111-113 (codon)	114-116 (codon)	117	118-120 (codon)
protein	55	56		57
AlignType	**PROTEIN**	**PROTEIN**	**SEQUENCE_ ERROR**	**PROTEIN**

The Alignment would have two **AlignmentElements**, one with the EST and one with the protein. Querying the Alignment with the **get_seq_region()** method would reveal the sequence regions listed above for each of the sequences. More importantly, the **get_align_type_by_column()** method for Column 22 would return a type **SEQUENCE_ERROR**, whereas for the other columns it would return a type **PROTEIN**. This way a program can confidently interpret the alignment. To indicate how important this information is, imagine if in Column 22 three bases were aligned. It would be ambiguous as to whether this indicated a protein insertion of a codon or a sequence error. The **AlignType** here provides this additional information.

The ability to associate a **CharacterAlignmentEncoder** with a more complex **Alignment** as this example is provides a way for clients to retrieve both the **Alignment** and a desired interpretation of the **Alignment** from the server, which facilitates writing alignment clients separately from actual alignment constructors. The **AlignmentEncoder**s provide a route for at least a character-based representation of the **Alignment** to be provided by the server, however complex the alignment method is. In this case, one might have one **AlignmentEncoder**, which provided the amino acids from the protein as three letter codes lined up with three bases from the EST. A different encoder might use one letter amino acid codes throughout, and not show the DNA sequence at all, choosing to encode the sequencing error with a special character.

2.1.17 Assembly

Assembly extends **Alignment**. **Assembly** contains no additional functionality. The technical domain is evolving rapidly and it's not clear what additional functionality will be necessary. However, the submitters believe it is important to establish the relationship between **Assembly** and **Alignment**.

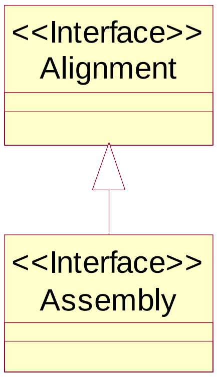


Figure 2-16 The Assembly Interface

```
interface Assembly : Alignment
{
};
```

2.1.18 SearchHit

The **SearchHit** valuetype provides a generic mechanism to return the results of some type of query against a collection of **BioSequence** objects. The **SearchHit** provides information about a particular sequence that was found and associated information for this hit relevant to this particular search, for an example, a score.

The **SearchHit** valuetype is used as a base class for the **SimilaritySearchHit**, which provides a specialization of the **SearchHit** for similarity searches

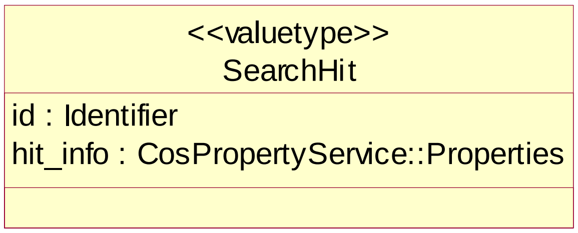


Figure 2-17 The SearchHit valuetype

SearchHit

```
valuetype SearchHit
{
    public Identifier                    id;
    public CosPropertyService::Properties  hit_info;
};
```

public Identifier id;	
Description:	The **Identifier** string identifies a sequence. It can be used with a **BioSequenceIdentifierResolver** to access the actual sequence.
Return value:	Returns an **Identifier** string. **id** shall not be empty.

public CosPropertyService::Properties hit_info;	
Description:	The **hit_info** provides additional information that is not contained in the **BioSequence** but is relevant from the perspective of the search. Common information would be the score in a similarity comparison, the statistical probability of the hit or the relevance of the hit in a text search. Content and type of information returned will vary with analysis type.
Return value:	Returns a **CosPropertyService::Properties**.

The following BLAST example illustrates the type of information that would be placed in **hit_info**. The example is taken from NCBI's BLAST help page. The associated alignment information is discussed below in the description of **SimilaritySearchHit**.

```
                                                              Smallest
                                                              Sum
                                                              High  Probability
Sequences producing High-scoring Segment Pairs:        Score  P(N)      N

sp|P05120|PAI2_HUMAN PLASMINOGEN ACTIVATOR INHIBITOR-2, P... 176  1.8e-65   4

[information deleted - ed.]

>sp|P05120|PAI2_HUMAN PLASMINOGEN ACTIVATOR INHIBITOR-2, PLACENTAL (PAI-2)
            (MONOCYTE ARG- SERPIN).
            Length = 415

 Score = 176 (80.2 bits), Expect = 1.8e-65, Sum P(4) = 1.8e-65
 Identities = 38/89 (42%), Positives = 50/89 (56%)
```

SearchHitList

typedef sequence<SearchHit> SearchHitList;	
Description:	Used to pass a set of **SearchHits**.

SearchHitIterator

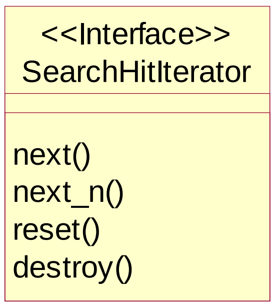


Figure 2-18 The SearchHitIterator Interface

```
interface SearchHitIterator
{
    boolean     next(out SearchHit hit)
                    raises(IteratorInvalid);
    boolean     next_n(in unsigned long how_many,
                        out SearchHitList hit_list)
                    raises(IteratorInvalid);
    void        reset();
    void        destroy();
};
```

boolean next(out SearchHit hit) raises(IteratorInvalid);	
Description:	The **next()** operation gets the next **SeachHit** in its out parameter **hit** and returns a boolean value. If the iterator is at the end of the set, it returns FALSE and sets the output **hit** parameter to null.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises IteratorInvalid if the iterator is no longer valid (e.g., the underlying collection has changed).

boolean next_n(in unsigned long how_many, out SearchHitList hit_list) raises(IteratorInvalid);	
Description:	**next_n()** returns **SearchHits** in the **SearchHitList** out parameter **hit_list**, containing at most the number specified in the first parameter (**how_many**) and returns a boolean value directly. When it is at the end of the set it returns FALSE and the **hit_list** parameter will have length zero. In all cases the length of **hit_list** will be the minimum of **how_many** and the number of elements remaining.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises IteratorInvalid if the iterator is no longer valid (e.g., the underlying collection has changed).

void reset();	
Description:	**reset()** sets the iterator to the start of the set.
Exceptions:	Raises CORBA::NO_IMPLEMENT with standard minor code 5 if the iterator cannot be reset (e.g., the iterator provides access to streaming data).

void destroy();	
Description:	**destroy()** frees the iterator object.

2.1.19 SimilaritySearchHit

The **SimilaritySearchHit** valuetype provides a specialization of the **SearchHit** valuetype for searches of **BioSequence** collections that are on the basis of similarity, such as BLAST, Fasta, or Smith-Waterman searches.

The member **alignment_list** is added to the **SearchHit** valuetype. This **AlignmentList** contains the details of a similarity search hit in the form of an **Alignment**. This list will frequently be of length one, but can be used to group all “local” hits pertaining to a single found sequence into one **SimilaritySearchHit**. That is, one **SimilaritySearchHit** may contain one “local match” or several “local matches” on one sequence. What is most appropriate depends on the analysis that was run to obtain the hits, and/or on the objective of a service; an implementation must document these semantics.

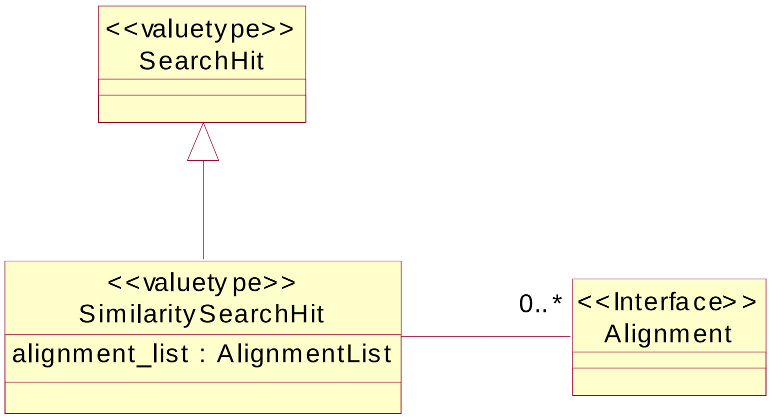


Figure 2-19 The SimilaritySearchHit valuetype

```
valuetype SimilaritySearchHit : SearchHit
{
	public AlignmentList  alignment_list;
};
```

public AlignmentList alignment_list;	
Description:	This attribute provides a list of **Alignments** that are associated with this hit. Not all hits may have alignments. In the **Alignments**, the sequence or object that was used as a query is the first **AlignmentElement** and the other objects (in most cases, just one) follow. If the **alignment_list** contains more than one **Alignment**, each **Alignment** shall involve the same two objects, i.e., only two **Identifiers** are used within one **SimilaritySearchHit**, one for the initial **query_sequence** (see **SearchHit**) and one for the current target sequence. Since each **Alignment** in a **SimilaritySearchHit** is a pairwise alignment, the result of the **Alignment's** **num_rows**() method shall be 2.
Return value:	Returns a list of **Alignments**.

The following BLAST example illustrates the alignment information that may be associated with a **SimilaritySearchHit**. The example is taken from NCBI's BLAST help page.

```
>sp|P05120|PAI2_HUMAN PLASMINOGEN ACTIVATOR INHIBITOR-2, PLACENTAL (PAI-2)
            (MONOCYTE ARG- SERPIN).
            Length = 415

 Score = 176 (80.2 bits), Expect = 1.8e-65, Sum P(4) = 1.8e-65
 Identities = 38/89 (42%), Positives = 50/89 (56%)

Query:     1 QIKDLLVSSSTDLDTTLVLVNAIYFKGMWKTAFNAEDTREMPFHVTKQESKPVQMMCMNN 60
                   +I +LL   S D DT +VLVNA+YFKG WKT F   +     PF V   +  PVQMM +
Sbjct:   180 KIPNLLPEGSVDGDTRMVLVNAVYFKGKWKTPFEKKLNGLYPFRVNSAQRTPVQMMYLRE 239


Query:    61 SFNVATLPAEKMKILELPFASGDLSMLVL 89
                      N+   +   K +ILELP+A      L+L
Sbjct:   240 KLNIGYIEDLKAQILELPYAGDVSMFLLL 268
```

SimilaritySearchHitList

typedef sequence<SimilaritySearchHit> SimilaritySearchHitList;	
Description:	Used to pass a set of **SimilaritySearchHits**.

This **AlignmentList** contains the details of a similarity search hit in the form of an **Alignment**. This list will frequently be of length one, but can be used to group all “local” hits pertaining to a single found sequence into one **SimilaritySearchHit**. That is, one **SimilaritySearchHit** may contain one “local match” or several “local matches” on one sequence. What is most appropriate depends on the analysis that was run to obtain the hits, and/or on the objective of a service; an implementation must document these semantics.

2.1.20 BioSequenceIdentifierResolver

The **BioSequenceIdentifierResolver** provides a mechanism to retrieve the actual **BioSequence** object from a collection search, using the **Identifier** string.

Implementors may want to consider multiply inheriting from **BioSequenceIdentifierResolver** interface with the optional **BioSequence** factories to provide sequence creation for an **Identifier**.

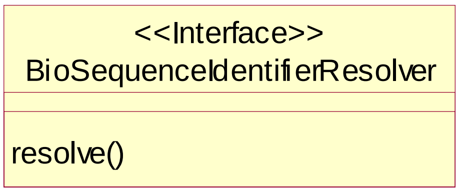


Figure 2-20 The BioSequenceIdentifierResolver interface

```
interface BioSequenceIdentifierResolver
{
    BioSequence resolve(in Identifier id)
        raises (IdentifierNotFound, IdentifierNotResolvable,
                IdentifierNotUnique);
};
```

BioSequence resolve(in Identifier id) raises (IdentifierNotFound, IdentifierNotResolvable, IdentifierNotUnique);	
Description:	The **resolve()** method provides the **BioSequence** for the particular **Identifier**.
Return value:	Returns a **BioSequence**.
Exceptions:	Raises IdentifierNotFound if the database and the identifier within the database can be resolved but the **Identifier** is not present. Raises IdentifierNotResolvable if the database and the identifier within the database cannot be resolved such that the **Identifier** cannot even be searched for. Raises IdentifierNotUnique if the **Identifier** specification is ambiguous and returns more than one object.

2.1.21 SearchResult

The **SearchResult** interface provides the complete results of a single search against a collection of **BioSequences**, including the individual hits and their associated scores and information about the search as whole. This interface is designed to represent results from both similarity queries on a database (such as BLAST, Fasta or Smith-Waterman) and text based searches on a database of **BioSequences**.

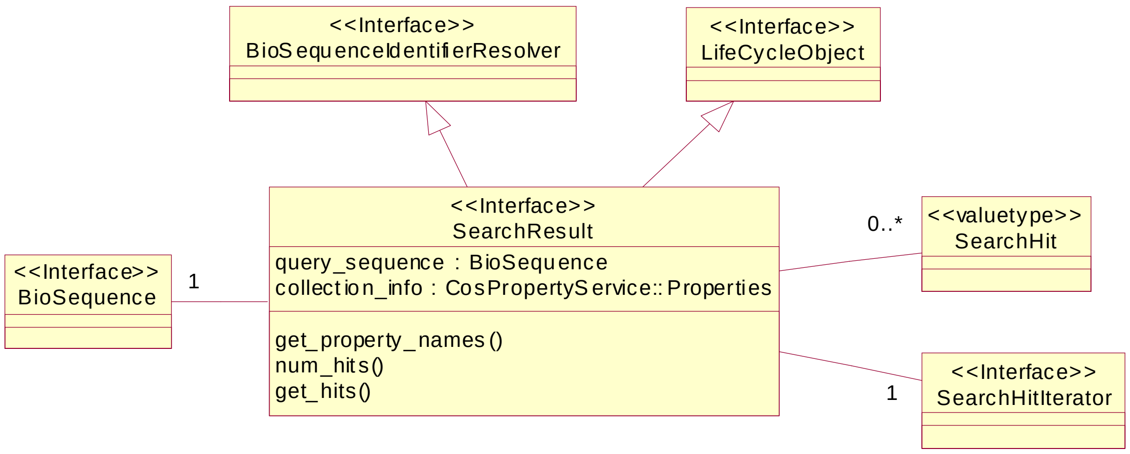


Figure 2-21 The SearchResult Interface

SearchResult

The **SearchResult** interface inherits from the **BioSequenceIdentifierResolver** to allow the retrieval of the actual **BioSequences** from the collection. It also inherits from **CosLifeCycle::LifeCycleObject** to allow management of its resources.

```
interface SearchResult :
      BioSequenceIdentifierResolver,
      CosLifeCycle::LifeCycleObject
{
      readonly attribute BioSequence  query_sequence;
      readonly attribute CosPropertyService::Properties  collection_info;
      StringList get_property_names();

      unsigned long  num_hits();

      SearchHitList  get_hits(
                  in unsigned long start,
                  in unsigned long how_many,
                  out SearchHitIterator the_rest)
            raises (IndexOutOfBounds);
};
```

readonly attribute BioSequence query_sequence;	
Description:	This attribute provides the query sequence that was used in this **SearchResult**. It may be null in the case of non similarity based searches.
Return value:	Returns a **BioSequence**.

readonly attribute CosPropertyService::Properties collection_info;	
Description:	The **collection_info** provides additional information that is not contained in the **SearchHits** but is relevant from the perspective of the search. Common information would be the database Identifier, the number of sequences in the database, and some statistical information about the search.
Return value:	Returns a **CosPropertyService::Properties**.

The following BLAST example illustrates the type of information that could be placed in **collection_info**. The example is taken from NCBI's BLAST help page.

```
BLASTP 1.4.6MP [13-Jun-94] [Build 13:58:36 Sep 22 1994]

Reference:  Altschul, Stephen F., Warren Gish, Webb Miller, Eugene W. Myers,
and David J. Lipman (1990).  Basic local alignment search tool.  J. Mol. Biol.
215:403-10.

Query =  pir|A01243|DXCH  232 Gene X protein - Chicken (fragment)
              (232 letters)

Database:  SWISS-PROT Release 29.0
                          38,303 sequences; 13,464,008 total letters.
Searching..................................................done


     Observed Numbers of Database Sequences Satisfying
     Various EXPECTation Thresholds (E parameter values)

       Histogram units:      = 31 Sequences     : less than 31 sequences

EXPECTation Threshold
(E parameter)
   |
   V   Observed Counts-->
 10000                 4863   1861 |============================================================
  6310                 3002     782 |=========================
  3980                 2220     812 |==========================
  2510                 1408     303 |=========
  1580                 1105     393 |============
  1000                 712      179 |=====
   631                 533      161 |=====
   398                 372       80 |==
   251                 292       73 |==
   158                 219       50 |=
   100                 169       32 |=
  63.1                 137       18 |:
  39.8                 119        9 |:
  25.1                 110        6 |:
  15.8                 104        9 |:
```

```
  >>>>>>>>>>>>>>>>>>>>>  Expect = 10.0, Observed = 95  <<<<<<<<<<<<<<<<<
  10.0              95          4 |:
  6.31              91          3 |:
  3.98              88          1 |:
  2.51              87          3 |:
  1.58              84          0 |
  1.00              84          2 |:

[SearchHit information deleted – ed.]

WARNING:  HSPs involving 86 database sequences were not reported due to the
         limiting value of parameter B = 9.


Parameters:
  V=15
  B=9
  H=1

  -ctxfactor=1.00
  E=10

  Query                                            -----  As Used  -----      -----  Computed  ----
  Frame                   MatID   Matrix name      Lambda      K      H       Lambda    K         H
  +0                      0       BLOSUM62         0.316       0.132  0.370   same      same     same

  Query
  Frame                MatID   Length  Eff.Length  E     S  W   T  X      E2  S2
  +0                   0       232     232         10.   57 3  11 22    0.22 33

Statistics:
  Query                          Expected         Observed           HSPs             HSPs
  Frame                MatID   High Score       High Score       Reportable    Reported
  +0                   0       62 (28.2 bits)  1191 (542.5 bits)         330                 24

  Query                        Neighborhd    Word      Excluded     Failed       Successful   Overlaps
  Frame                MatID       Words      Hits           Hits    Extensions   Extensions   Excluded
  +0                   0           4988    5661199     1146395     4504598        10187           13

  Database:  SWISS-PROT Release 29.0
   Release date:  June 1994
   Posted date:  1:29 PM EDT Jul 28, 1994
  # of letters in database:  13,464,008
  # of sequences in database:  38,303
  # of database sequences satisfying E:  95
  No. of states in DFA:  561 (55 KB)
  Total size of DFA:  110 KB (128 KB)
  Time to generate neighborhood:  0.03u 0.01s 0.04t  Real: 00:00:00
  No. of processors used:  8
  Time to search database:  32.27u 0.78s 33.05t  Real: 00:00:04
  Total cpu time:  32.33u 0.91s 33.24t  Real: 00:00:05

WARNINGS ISSUED:  2
```

StringList get_property_names();	
Description:	The names of the **hit_info** properties in **SearchHit** are available here so that clients have access to them before processing the list of **SearchHits**.
Return value:	Returns a **StringList**.

unsigned long num_hits();	
Description:	Provides the number of hits in this **SearchResult**.
Return value:	Returns an **unsigned long**.

SearchHitList get_hits(in unsigned long start, in unsigned long how_many, out SearchHitIterator the_rest) raises (IndexOutOfBounds);	
Description:	Uses the list/iterator hybrid to provide access to the actual **SearchHits**, which could be **SimilaritySearchHits**. A list of no more than **how_many** hits starting at **start** is returned as the direct result. The remaining elements, if any, are available through the iterator returned in the **out** parameter.
Return value:	Returns a **SearchHitList**.
Exceptions:	Raises IndexOutOfBounds if the index is less than 1 or greater than the number of hits in the **SearchResult**. This upper limit is returned by **num_hits()**.

The method **get_hits**() returns a list of **SearchHits**. If this concerns **SimilaritySearchHits**, then these elements will in turn contain a list of **Alignments**. The latter can be used to group all “local” hits pertaining to a single found sequence into one **SimilaritySearchHit**. That is, one **SimilaritySearchHit** may contain one “local match” or several “local matche”' on one sequence. What is most appropriate depends on the analysis that was run to obtain the hits, and/or on the objective of a service. An implementation must document these semantics.

2.1.22 *AnnotationFactory (Optional)*

AnnotationFactory provides a means of creating new **Annotation** and **SeqAnnotation** objects. This permits a clean separation of factory issues from the **Annotation** objects themselves. **Annotations** are created via the factory method

create_annotation(), which accepts all of the components. Similarly, **SeqAnnotations** are created via the factory method **create_seq_annotation()**, which accepts all of the components.

AnnotationFactory is an optional compliance point of this specification.

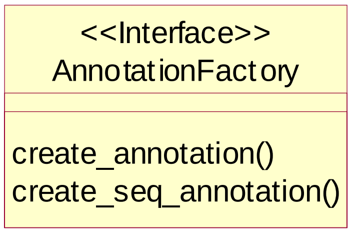


Figure 2-22 The AnnotationFactory Interface

```
interface AnnotationFactory
{
    Annotation create_annotation(
            in string name,
            in any value,
            in Basis the_basis,
            in CosPropertyService::Properties qualifiers);

    SeqAnnotation create_seq_annotation(
            in string name,
            in any value,
            in Basis the_basis,
            in CosPropertyService::Properties qualifiers,
            in SeqRegion seq_region)
        raises(SeqRegionInvalid);
};
```

Annotation create_annotation(in string name, in any value, in Basis the_basis, in CosPropertyService::Properties qualifiers);	
Description:	The **create_annotation()** operation creates an **Annotation** and populates it with the supplied attributes. No error checking is performed.
Return value:	Returns an **Annotation** with the appropriate content.

SeqAnnotation create_seq_annotation(in string name, in any value, in Basis the_basis, in CosPropertyService::Properties qualifiers, in SeqRegion seq_region) raises(SeqRegionInvalid);	
Description:	The **create_seq_annotation()** operation creates a **SeqAnnotation** and populates it with the supplied attributes. No error checking is performed.
Return value:	Returns a **SeqAnnotation** with the appropriate content.
Exceptions:	Raises SeqRegionInvalid if the **seq_region** parameter is deemed invalid. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

2.1.23 *BioSequence factories (Optional)*

Sequence factories permit a clean separation of object vending from **BioSequence** data model issues. **BioSequence** factories are an optional compliance point of this submission.

BioSequence factories provide a means of creating new **NucleotideSequence** and **AminoAcidSequence** objects. Sequences are created via the factory method **create_sequence()**, which accepts all of the components.

Implementors may want to consider mixing in the **BioSequenceIdentifierResolver** interface to provide sequence creation for an **Identifier**.

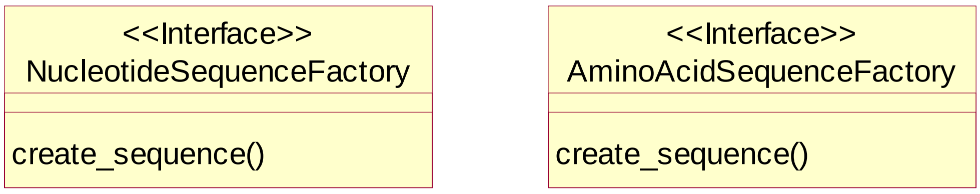


Figure 2-23 The BioSequence Factories

SeqAnnotationOutOfBounds

```
exception SeqAnnotationOutOfBounds
{
    SeqAnnotation invalid;
    Interval valid;
};
```

Description:	The SeqAnnotationOutOfBounds exception is raised if a **SeqAnnotation**'s **SeqRegion** has a **start** less than 1 or if its **start+length-1** is greater than the length of the **BioSequence**. The exception is also raised if a nested sub-region of a **CompositeSeqRegion** is invalid. If a **BioSequence** represents circular DNA, then this exception should be raised if the **Interval**'s **start** is less than 1 or greater than the **length** of the **BioSequence**, or if its **length** is greater than that of the **BioSequence**.
Return value:	Returns the invalid SeqAnnotation and the valid **Interval**. The valid **Interval** has **start** equal to 1 and **length** equal to the length of the **BioSequence**, the largest allowed **Interval**.

NucleotideSequenceFactory

NucleotideSequenceFactory provides a means of creating new **NucleotideSequences**. **NucleotideSequenceFactory** is an optional compliance point of this specification.

```
interface NucleotideSequenceFactory
{
    NucleotideSequence create_sequence(
            in string name,
            in Identifier id,
            in string description,
            in string residues,
            in Basis the_basis,
            in boolean circular,
            in AnnotationList annotations)
        raises (InvalidResidue, SeqAnnotationOutOfBounds, SeqRegionInvalid);
};
```

NucleotideSequence create_sequence(in string name, in Identifier id, in string description, in string residues, in Basis the_basis, in boolean circular, in AnnotationList annotations) raises (InvalidResidue, SeqAnnotationOutOfBounds, SeqRegionInvalid);	
Description:	The **create_sequence()** operation creates a **NucleotideSequence** and populates it with the supplied attributes. No error checking is performed except on the residues, which must be valid IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) single letter codes. The residues need not be upper-case. **BioSequenceIdentifierResolver** can be mixed in to provide lookup based on sequence ID.
Return value:	Returns a **NucleotideSequence** with the appropriate content.
Exceptions:	Raises InvalidResidue if the string of residues is inconsistent with the IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) single letter codes. Note that residue is interpreted to mean base here (see Glossary). Raises SeqAnnotationOutOfBounds if **annotations** contains a **SeqAnnotation** whose **seq_region** is out of bounds for this **BioSequence**. Raises SeqRegionInvalid if the **annotations** parameter contains **SeqAnnotation** having a **seq_region** attribute that is deemed invalid. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

AminoAcidSequenceFactory

AminoAcidSequenceFactory provides a means of creating new **AminoAcidSequences**. **AminoAcidSequenceFactory** is an optional compliance point of this specification.

```
interface AminoAcidSequenceFactory
{
    AminoAcidSequence create_sequence(
                in string name,
                in Identifier id,
                in string description,
                in string residues,
                in Basis the_basis,
                in AnnotationList annotations)
            raises (InvalidResidue, SeqAnnotationOutOfBounds, SeqRegionInvalid);
};
```

AminoAcidSequence create_sequence(in string name, in Identifier id, in string description, in string residues, in Basis the_basis, in AnnotationList annotations) raises (InvalidResidue, SeqAnnotationOutOfBounds, SeqRegionInvalid);	
Description:	The **create_sequence()** operation creates an **AminoAcidSequence** and populates it with the supplied attributes. No error checking is performed except on the residues, which must be valid IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) single letter codes. The residues need not be upper-case. **BioSequenceIdentifierResolver** can be mixed in to provide lookup based on sequence ID.
Return value:	Returns a **AminoAcidSequence** with the appropriate content.
Exceptions:	Raises InvalidResidue if the string of residues is inconsistent with the IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) single letter codes. Raises SeqAnnotationOutOfBounds if annotations contains a **SeqAnnotation** whose **seq_region** is out of bounds for this **BioSequence**. Raises SeqRegionInvalid if the **annotations** parameter contains **SeqAnnotation** having a **seq_region** attribute that is deemed invalid. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

2.1.24 BioSequence iterators (Optional)

Iterator specifications are defined for iterating over a set of **BioSequence**, **NucleotideSequence**, or **AminoAcidSequence** objects. **NucleicAcidIterator** and **AminoAcidIterator** are specialized versions of **BioSequenceIterator** having the same operations but with signatures specialized for the corresponding **BioSequence** sub-types. **BioSequenceIterator** and **BioSequenceList** may contain both **NucleotideSequences** and **AminoAcidSequences**. Homogeneity in the sequence types of iterators and lists can be achieved using the specialized versions.

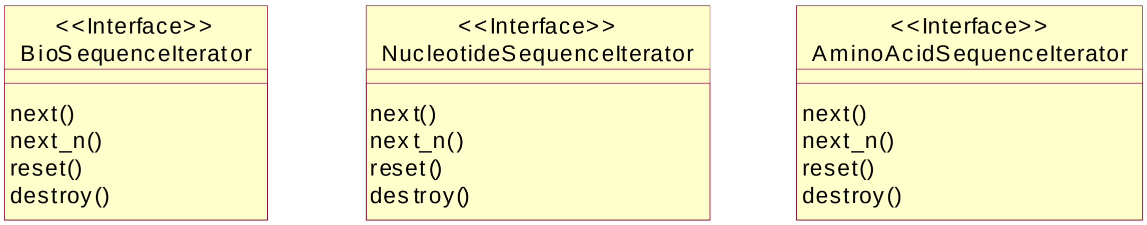


Figure 2-24 The BioSequence Iterators

BioSequenceIterator

BioSequenceIterator provides a strongly typed iterator for **BioSequences**.

```
interface BioSequenceIterator
{
      boolean      next(out BioSequence seq)
                       raises(IteratorInvalid);
      boolean      next_n(in unsigned long how_many,
                               out BioSequenceList seqs)
                       raises(IteratorInvalid);
      void         reset();
      void         destroy();
};
```

boolean next(out BioSequence seq) raises(IteratorInvalid);	
Description:	The **next()** operation gets the next **BioSequence** in its out parameter **seq** and returns a boolean value. If the iterator is at the end of the set, it returns FALSE and sets the output **seq** parameter to null.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises IteratorInvalid if the iterator is no longer valid (e.g., the underlying collection has changed).

boolean next_n(in unsigned long how_many, out BioSequenceList seqs) raises(IteratorInvalid);	
Description:	**next_n()** returns **BioSequences** in the **BioSequenceList** out parameter **seqs**, containing at most the number specified in the first parameter (**how_many**) and returns a boolean value directly. When it is at the end of the sequence set it returns FALSE and the **seqs** parameter will have length zero. In all cases the length of **seqs** will be the minimum of **how_many** and the number of sequences remaining.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises IteratorInvalid if the iterator is no longer valid (e.g., the underlying collection has changed).

void reset();	
Description:	**reset()** sets the iterator to the start of the set.
Exceptions:	Raises CORBA::NO_IMPLEMENT with standard minor code 5 if the iterator cannot be reset (e.g., the iterator provides access to streaming data).

void destroy();	
Description:	**destroy()** frees the iterator object.

NucleotideSequenceIterator

NucleotideSequenceIterator provides a strongly typed iterator for **NucleotideSequences**.

```
interface NucleotideSequenceIterator
{
    boolean     next(out NucleotideSequence seq)
                    raises(IteratorInvalid);
    boolean     next_n(in unsigned long how_many,
                        out NucleotideSequenceList seqs)
                    raises(IteratorInvalid);
    void        reset();
    void        destroy();
};
```

boolean next(out NucleotideSequence seq) raises(IteratorInvalid);	
Description:	The **next()** operation gets the next **NucleotideSequence** in its out parameter **seq** and returns a boolean value. If the iterator is at the end of the set, it returns FALSE and sets the output **seq** parameter to null.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises IteratorInvalid if the iterator is no longer valid (e.g., the underlying collection has changed).

boolean next_n(in unsigned long how_many, out NucleotideSequenceList seqs) raises(IteratorInvalid);	
Description:	**next_n()** returns **NucleotideSequences** in the **NucleotideSequenceList** out parameter **seqs**, containing at most the number specified in the first parameter (**how_many**) and returns a boolean value directly. When it is at the end of the sequence set it returns FALSE and the **seqs** parameter will have length zero. In all cases the length of **seqs** will be the minimum of **how_many** and the number of sequences remaining.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises IteratorInvalid if the iterator is no longer valid (e.g., the underlying collection has changed).

void reset();	
Description:	**reset()** sets the iterator to the start of the set.
Exceptions:	Raises CORBA::NO_IMPLEMENT with standard minor code 5 if the iterator cannot be reset (e.g., the iterator provides access to streaming data).

void destroy();	
Description:	**destroy()** frees the iterator object.

AminoAcidSequenceIterator

AminoAcidSequenceIterator provides a strongly typed iterator for **AminoAcidSequences**.

```
interface AminoAcidSequenceIterator
{
    boolean     next(out AminoAcidSequence seq)
                    raises(IteratorInvalid);
    boolean     next_n(in unsigned long how_many,
                        out AminoAcidSequenceList seqs)
                    raises(IteratorInvalid);
    void        reset();
    void        destroy();
};
```

boolean next(out AminoAcidSequence seq) raises(IteratorInvalid);	
Description:	The **next()** operation gets the next **AminoAcidSequence** in its out parameter **seq** and returns a boolean value. If the iterator is at the end of the set, it returns FALSE and sets the output **seq** parameter to null.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises IteratorInvalid if the iterator is no longer valid (e.g., the underlying collection has changed).

boolean next_n(in unsigned long how_many, out AminoAcidSequenceList seqs) raises(IteratorInvalid);	
Description:	**next_n()** returns **AminoAcidSequences** in the **AminoAcidSequenceList** out parameter **seqs**, containing at most the number specified in the first parameter (**how_many**) and returns a boolean value directly. When it is at the end of the sequence set it returns FALSE and the **seqs** parameter will have length zero. In all cases the length of **seqs** will be the minimum of **how_many** and the number of sequences remaining.
Return value:	Returns FALSE if the iterator is at the end of the set and TRUE otherwise.
Exceptions:	Raises IteratorInvalid if the iterator is no longer valid (e.g., the underlying collection has changed).

void reset();	
Description:	**reset()** sets the iterator to the start of the set.
Exceptions:	Raises CORBA::NO_IMPLEMENT with standard minor code 5 if the iterator cannot be reset (e.g., the iterator provides access to streaming data).

void destroy();	
Description:	**destroy()** frees the iterator object.

2.1.25 GeneticCodeFactory (Optional)

GeneticCodeFactory provides a means of creating new **GeneticCodes**. **GeneticCodeFactory** is an optional compliance point of this specification.

InvalidGeneticCodeName

```
exception InvalidGeneticCodeName
{
    string invalid_name;
};
```

Description:	The **InvalidGeneticCodeName** exception is raised when an invalid **GeneticCodeName** is passed to **GeneticCodeFactory's create_genetic_code()**.
Return value:	Returns a **string** containing the invalid name.

GeneticCodeFactory

The **GeneticCodeFactory** interface defines a set of **const GeneticCodeName** strings that list the set of currently known genetic codes. The **genetic_code_names** attribute provides access to the supported **GeneticCodeNames**. **create_genetic_code()** creates the appropriate **GeneticCode**. **Codings** for the **GeneticCodeNames** listed below can be found in Appendix B.

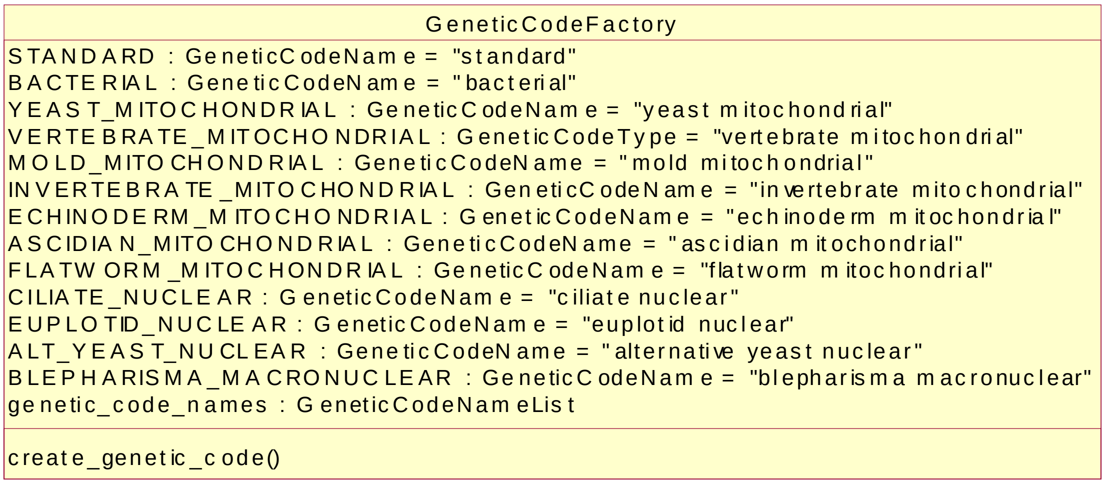

GeneticCodeFactory
STANDARD : GeneticCodeName = "standard"
BACTERIAL : GeneticCodeName = "bacterial"
YEAST_MITOCHONDRIAL : GeneticCodeName = "yeast mitochondrial"
VERTEBRATE_MITOCHONDRIAL : GeneticCodeType = "vertebrate mitochondrial"
MOLD_MITOCHONDRIAL : GeneticCodeName = "mold mitochondrial"
INVERTEBRATE_MITOCHONDRIAL : GeneticCodeName = "invertebrate mitochondrial"
ECHINODERM_MITOCHONDRIAL : GeneticCodeName = "echinoderm mitochondrial"
ASCIDIAN_MITOCHONDRIAL : GeneticCodeName = "ascidian mitochondrial"
FLATWORM_MITOCHONDRIAL : GeneticCodeName = "flatworm mitochondrial"
CILIATE_NUCLEAR : GeneticCodeName = "ciliate nuclear"
EUPLOTID_NUCLEAR : GeneticCodeName = "euplotid nuclear"
ALT_YEAST_NUCLEAR : GeneticCodeName = "alternative yeast nuclear"
BLEPHARISMA_MACRONUCLEAR : GeneticCodeName = "blepharisma macronuclear"
genetic_code_names : GeneticCodeNameList
create_genetic_code()

Figure 2-25 The GeneticCodeFactory Interface

```
interface GeneticCodeFactory
{
    const GeneticCodeName  STANDARD                      = "standard";
    const GeneticCodeName  BACTERIAL                     = "bacterial";
    const GeneticCodeName  YEAST_MITOCHONDRIAL           = "yeast mitochondrial";
    const GeneticCodeName  VERTEBRATE_MITOCHONDRIAL      = "vertebrate mitochondrial";
    const GeneticCodeName  MOLD_MITOCHONDRIAL            = "mold mitochondrial";
    const GeneticCodeName  INVERTEBRATE_MITOCHONDRIAL    = "invertebrate mitochondrial";
    const GeneticCodeName  ECHINODERM_MITOCHONDRIAL      = "echinoderm mitochondrial";
    const GeneticCodeName  ASCIDIAN_MITOCHONDRIAL        = "ascidian mitochondrial";
    const GeneticCodeName  FLATWORM_MITOCHONDRIAL        = "flatworm mitochondrial";
    const GeneticCodeName  CILIATE_NUCLEAR               = "ciliate nuclear";
    const GeneticCodeName  EUPLOTID_NUCLEAR              = "euplotid nuclear";
    const GeneticCodeName  ALT_YEAST_NUCLEAR             = "alternative yeast nuclear";
    const GeneticCodeName  BLEPHARISMA_MACRONUCLEAR      = "blepharisma macronuclear";

    readonly attribute GeneticCodeNameList genetic_code_names;
    GeneticCode create_genetic_code(in GeneticCodeName name)
        raises(InvalidGeneticCodeName);
};
```

const GeneticCodeName	**STANDARD**	**= "standard";**
const GeneticCodeName	**BACTERIAL**	**= "bacterial";**
const GeneticCodeName	**YEAST_MITOCHONDRIAL**	**= "yeast mitochondrial";**
const GeneticCodeName	**VERTEBRATE_MITOCHONDRIAL**	**= "vertebrate mitochondrial";**
const GeneticCodeName	**MOLD_MITOCHONDRIAL**	**= "mold mitochondrial";**
const GeneticCodeName	**INVERTEBRATE_MITOCHONDRIA**	**= "invertebrate mitochondrial";**
const GeneticCodeName	**ECHINODERM_MITOCHONDRIAL**	**= "echinoderm mitochondrial";**
const GeneticCodeName	**ASCIDIAN_MITOCHONDRIAL**	**= "ascidian mitochondrial";**
const GeneticCodeName	**FLATWORM_MITOCHONDRIAL**	**= "flatworm mitochondrial";**
const GeneticCodeName	**CILIATE_NUCLEAR**	**= "ciliate nuclear";**
const GeneticCodeName	**EUPLOTID_NUCLEAR**	**= "euplotid nuclear";**
const GeneticCodeName	**ALT_YEAST_NUCLEAR**	**= "alternative yeast nuclear";**
const GeneticCodeName	**BLEPHARISMA_MACRONUCLEAR**	**= "blepharisma macronuclear";**
Description:	The **GeneticCodeFactory** interface defines a set of **const GeneticCodeName** strings that list the set of currently known genetic codes. The **GeneticCodeName** defines the particular **Coding** that is used to convert **Codons** into **Residues** so one need only specify the **GeneticCodeName** when creating a **GeneticCode** object from one of the known types. **Codings** for the **GeneticCodeNames** listed above can be found in Appendix B.	

readonly attribute GeneticCodeNameList genetic_code_names;	
Description:	The **genetic_code_names** attribute provides access to the supported **GeneticCodeNames**.
Return value:	Returns a **GeneticCodeNameList**.

GeneticCode create_genetic_code(in GeneticCodeName name) raises(InvalidGeneticCodeName);	
Description:	**create_genetic_code()** creates the appropriate **GeneticCode** corresponding to the **GeneticCodeName**. **Codings** for the **GeneticCodeNames** listed above can be found in Appendix B.
Return value:	Returns a **GeneticCode**.
Exceptions:	Raises InvalidGeneticCodeName if the **GeneticCodeName** is not supported (i.e., returned by the **genetic_code_names** attribute).

2.1.26 CharacterAlignmentEncoder (Optional)

The **CharacterAlignmentEncoder** and its specialization **SingleCharacterAlignmentEncoder** are optional parts of the specification that facilitate the representation of the **Alignment** for thin clients. It is important that these interfaces have a proposed standard, as it will allow clients that do not want to investigate **Alignments** directly to get useful information for passing on to a user or to another, text format based application.

A **CharacterAlignmentEncoder**'s role is to produce string text similar to that in Table 2-4 on page 2-43, with columns of text indicating the correspondences and the row indicating each sequence. The exact format isn't specified or standardized. The factory that makes the encoder will govern the precise nature of the encoding, such as what pad character is used. The **CharacterAlignmentEncoder** might have more than one character per column, allowing the transmission of three-letter amino acid code or more than one base of DNA sequence in a single column. To allow the client to format the resulting data, **max_column_width()** returns the maximum length of characters in a column. Rows and columns are numbered starting at 1.

The **Alignment** and the **CharacterAlignmentEncoder** interfaces work well for both view-based clients and programmatic clients. The interfaces provide viewing clients with an easy, low cost route of gathering the alignment data and displaying it to the user. The coordinate system of the string encoded alignment maps to the underlying alignment, allowing the client to retrieve specific regions of the alignment of interest. Since the **Interval** valuetype can be used to retrieve only portions of the **BioSequences**, these very complex objects can remain on the server, with the clients displaying only portions of interest to the user. For programmatic clients wanting to use the alignment as the basis of further analysis, the **Alignment** interface provides a mapping system of moving from one sequence to another sequence via the alignment.

CharacterAlignmentEncoder is an optional compliance point of this specification.

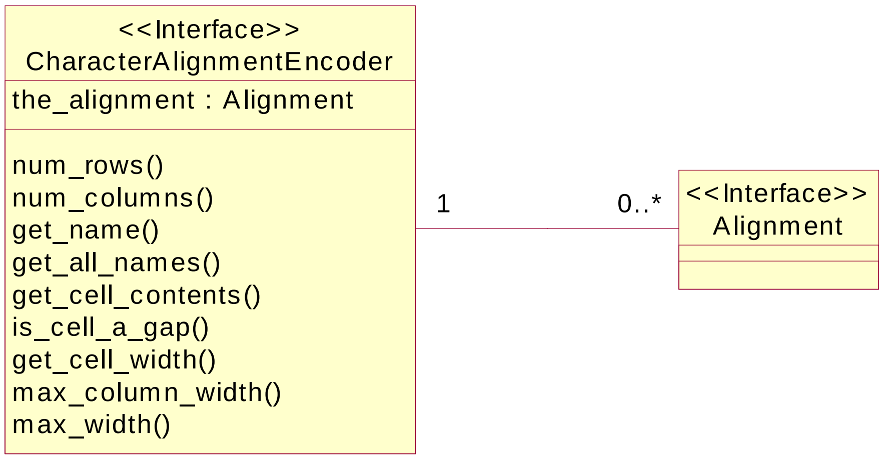


Figure 2-26 The CharacterAlignmentEncoder Interface

```
interface CharacterAlignmentEncoder
{
    readonly attribute Alignment the_alignment;
    unsigned long  num_rows();                    // number of aligned
                                                  // objects. Delegate
    unsigned long  num_columns();                 // Delegate to Alignment

    string     get_name(in unsigned long row)     // first object is in row
                     raises(IndexOutOfBounds);    // one etc...
    StringList get_all_names();                   // all the Names

    string     get_cell_contents(in unsigned long row, in unsigned long col)
                     raises(IndexOutOfBounds);
    boolean    is_cell_a_gap(in unsigned long row, in unsigned long col)
                     raises(IndexOutOfBounds);
    unsigned long  get_cell_width(in unsigned long row, in unsigned long col)
                     raises(IndexOutOfBounds);
    unsigned long  max_column_width(in unsigned long col)
                     raises(IndexOutOfBounds);
    unsigned long  max_width();
};
```

readonly attribute Alignment the_alignment;	
Description:	Provides access to the underlying **Alignment**.
Return value:	Returns an **Alignment**.

unsigned long num_rows();	
Description:	Provides access to the number of rows (**AlignmentElements**) in this **Alignment**. The return value of **num_rows()** is the same as that of the **Alignment**'s **num_rows()**.
Return value:	Returns an **unsigned long**.

unsigned long num_columns();	
Description:	Provides access to the total number of correspondences in this **Alignment**. The return value of **num_columns()** is the same as that of the **Alignment**'s **num_columns()**.
Return value:	Returns an **unsigned long**.

string get_name(in unsigned long row) raises(IndexOutOfBounds);	
Description:	Provides access to the name associated with the **AlignmentElement** referenced by **row**.
Return value:	Returns a **string**.
Exceptions:	Raises IndexOutOfBounds if **row** is less than 1 or greater than the number of rows. This upper limit is returned by **num_rows()**.

StringList get_all_names();	
Description:	Provides access to the names associated with each of the **AlignmentElements**.
Return value:	Returns a **StringList**, one **string** per **AlignmentElement**.

string get_cell_contents(in unsigned long row, in unsigned long col) raises(IndexOutOfBounds);	
Description:	Provides access to the string associated with a single cell. The cell corresponds to the correspondence **col** in the **AlignmentElement** referenced by **row**.
Return value:	Returns a **string**.
Exceptions:	Raises IndexOutOfBounds if **row** is less than 1 or greater than the number of rows. This upper limit is returned by **num_rows()**. Also raises IndexOutOfBounds if **col** is less than 1 or greater than the number of columns. This upper limit is returned by **num_columns()**.

boolean is_cell_a_gap(in unsigned long row, in unsigned long col) raises(IndexOutOfBounds);	
Description:	Indicates if a single cell represents a gap in the alignment. The cell corresponds to the correspondence **col** in the **AlignmentElement** referenced by **row**.
Return value:	Returns a **boolean**.
Exceptions:	Raises IndexOutOfBounds if **row** is less than 1 or greater than the number of rows. This upper limit is returned by **num_rows()**. Also raises IndexOutOfBounds if **col** is less than 1 or greater than the number of columns. This upper limit is returned by **num_columns()**.

unsigned long get_cell_width (in unsigned long row, in unsigned long col) raises(IndexOutOfBounds);	
Description:	To allow the client to format the resulting data, **get_cell_width()** returns the width of a single cell. The cell corresponds to the correspondence **col** in the **AlignmentElement** referenced by **row**.
Return value:	Returns an **unsigned long**.
Exceptions:	Raises IndexOutOfBounds if **row** is less than 1 or greater than the number of rows. This upper limit is returned by **num_rows()**. Also raises IndexOutOfBounds if **col** is less than 1 or greater than the number of columns. This upper limit is returned by **num_columns()**.

unsigned long max_column_width(in unsigned long col) raises(IndexOutOfBounds);	
Description:	To allow the client to format the resulting data, **max_column_width()** returns the maximum length of characters in a column defined by **col**.
Return value:	Returns an **unsigned long**.
Exceptions:	Raises IndexOutOfBounds if **col** is less than 1 or greater than the number of columns. This upper limit is returned by **num_columns()**.

unsigned long max_width();	
Description:	To allow the client to format the resulting data, **max_width()** returns the maximum length of characters in the widest column.
Return value:	Returns an **unsigned long**.

2.1.27 SingleCharacterAlignmentEncoder (Optional)

A **SingleCharacterAlignmentEncoder** is one in which each correspondence is guaranteed to have only a single character for all **AlignmentElements**. Therefore, more bulk transport mechanisms can be employed, using strings to get rows of the **Alignment** or the entire **Alignment** as a block of text.

SingleCharacterAlignmentEncoder is an optional compliance point of this specification.

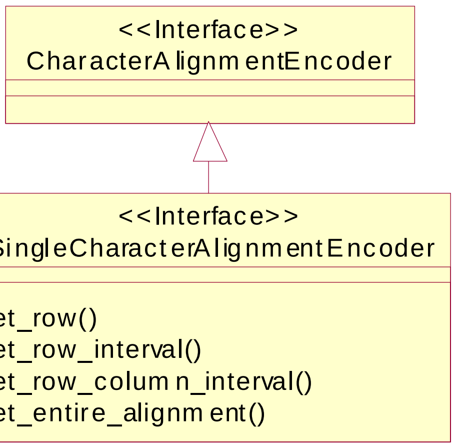

<<Interface>>
CharacterAlignmentEncoder
<<Interface>>
SingleCharacterAlignmentEncoder
get_row()
get_row_interval()
get_row_column_interval()
get_entire_alignment()

Figure 2-27 The SingleCharacterAlignmentEncoder Interface

```
interface SingleCharacterAlignmentEncoder : CharacterAlignmentEncoder
{
    string          get_row(in unsigned long row)
                        raises(IndexOutOfBounds);
    string          get_row_interval(in unsigned long row, in Interval cols)
                        raises(IndexOutOfBounds, IntervalOutOfBounds,
                            SeqRegionInvalid);
    StringList      get_row_column_interval(in Interval rows, in Interval cols)
                        raises(IntervalOutOfBounds, SeqRegionInvalid);
    StringList      get_entire_alignment();          // probably the most common!
};
```

string get_row(in unsigned long row) **raises(IndexOutOfBounds);**	
Description:	Provides the text for all of a single **AlignmentElement** as a **string**. **row** identifies the **AlignmentElement**. There is one character per cell.
Return value:	Returns a **string**.
Exceptions:	Raises IndexOutOfBounds if **row** is less than 1 or greater than the number of rows. This upper limit is returned by **num_rows()**, inherited from **CharacterAlignmentEncoder**.

string get_row_interval(in unsigned long row, in Interval cols) **raises(IndexOutOfBounds, IntervalOutOfBounds, SeqRegionInvalid);**	
Description:	Provides the text for part of a single **AlignmentElement** as a **string**. **row** identifies the **AlignmentElement**. **cols** allows a subset of the correspondences to be referenced. There is one character per cell.
Return value:	Returns a **string**.
Exceptions:	Raises IndexOutOfBounds if **row** is less than 1 or greater than the number of rows. This upper limit is returned by **num_rows**(), inherited from **CharacterAlignmentEncoder**. Raises IntervalOutOfBounds if **cols' start** is less than 1 or **start+length-1** is greater than the number of columns. This upper limit is returned by **num_cols**(), inherited from **CharacterAlignmentEncoder**. Raises SeqRegionInvalid if the interval is an invalid **SeqRegion**. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

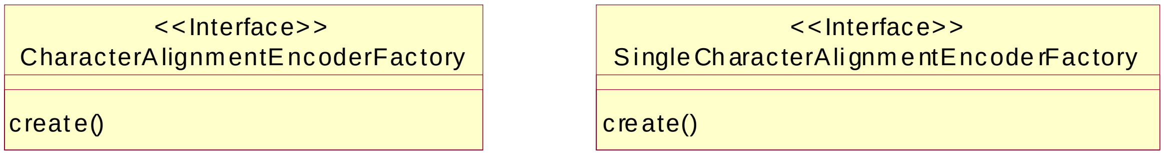
StringList get_row_column_interval(in Interval rows, in Interval cols) raises(IntervalOutOfBounds, SeqRegionInvalid);	
Description:	Provides the sub-block of text for the portion of the **Alignment** defined by **rows** and **cols** Intervals as an array of **strings**. **rows** allows a subset of the **AlignmentElements** to be referenced. **cols** allows a subset of the correspondences to be referenced. There is one character per cell.
Return value:	Returns a **StringList**, one **string** per row.
Exceptions:	Raises IntervalOutOfBounds if **rows**' **start** is less than 1 or **start+length-1** is greater than the number of rows. This upper limit is returned by **num_rows()**, inherited from **CharacterAlignmentEncoder**. Also raises IntervalOutOfBounds if **cols**' **start** is less than 1 or **start+length-1** is greater than the number of columns. This upper limit is returned by **num_cols()**, inherited from **CharacterAlignmentEncoder**. Raises SeqRegionInvalid if the interval is an invalid **SeqRegion**. Examples include an incorrect **StrandType**, or an invalid **CompositeSeqRegion** (e.g., one that has a wrong **SeqRegionOperator** or contains overlaps or circularities).

StringList get_entire_alignment();	
Description:	Provides the block of text for the entire **Alignment** as an array of **strings**. There is one character per cell.
Return value:	Returns a **StringList**, one **string** per row.

2.1.28 AlignmentEncoder factories (Optional)

AlignmentEncoder factories provide a means of creating new **CharacterAlignmentEncoder** and **SingleCharacterAlignmentEncoder** objects. This permits a clean separation of factory issues from the **AlignmentEncoder** objects themselves.

AlignmentEncoder factories are an optional compliance point of this specification.



<<Interface>>
CharacterAlignmentEncoderFactory
create()
<<Interface>>
SingleCharacterAlignmentEncoderFactory
create()

Figure 2-28 The AlignmentEncoder factories

CannotEncodeAlignment

exception CannotEncodeAlignment { string reason; };	
Description:	The CannotEncodeAlignment exception is raised if an **AlignmentEncoder** can not be created for this **Alignment**.
Return value:	Returns a **string** containing the reason the **AlignmentEncoder** could not be created for this **Alignment**.

CharacterAlignmentEncoderFactory

CharacterAlignmentEncoderFactory provides a means of creating new **CharacterAlignmentEncoders** for an **Alignment**. **CharacterAlignmentEncoderFactory** is an optional compliance point of this specification.

```
interface CharacterAlignmentEncoderFactory
{
    CharacterAlignmentEncoder create(in Alignment the_alignment)
        raises(CannotEncodeAlignment);
};
```

CharacterAlignmentEncoder create(in Alignment the_alignment) raises(CannotEncodeAlignment);	
Description:	The **create()** operation creates a **CharacterAlignmentEncoder** for the given **Alignment**.
Return value:	Returns a **CharacterAlignmentEncoder**.
Exceptions:	Raises CannotEncodeAlignment if a **CharacterAlignmentEncoder** cannot be created for this **Alignment**.

SingleCharacterAlignmentEncoderFactory

SingleCharacterAlignmentEncoderFactory provides a means of creating new **SingleCharacterAlignmentEncoders** for an **Alignment**. **SingleCharacterAlignmentEncoderFactory** is an optional compliance point of this specification.

```
interface SingleCharacterAlignmentEncoderFactory
{
    SingleCharacterAlignmentEncoder create(in Alignment the_alignment)
        raises(CannotEncodeAlignment);
};
```

SingleCharacterAlignmentEncoder create(in Alignment the_alignment) raises(CannotEncodeAlignment);	
Description:	The **create**() operation creates a **SingleCharacterAlignmentEncoder** for the given **Alignment**.
Return value:	Returns a **SingleCharacterAlignmentEncoder**.
Exceptions:	Raises CannotEncodeAlignment if a **SingleCharacterAlignmentEncoder** cannot be created for this **Alignment**.

2.2 Module DsLSRAnalysis

The **DsLSRAnalysis** module defines the component interfaces for supporting sequence analysis through a generic analysis design. The module encapsulates the required elements for analysis. It provides the means to interrogate analyses inputs, output and functionality. An analysis can be executed asynchronously as well as synchronously based on the client invocation. Executing analyses can be monitored by subscribing to an event channel or polling for state.

The Client is responsible for:

- determining which Biomolecular Sequence Analysis (BSA) analysis tool (e.g., BLAST, Smith-Waterman, etc.) it wants to employ;
- locating an **AnalysisService** that represent the BSA analysis tool;
- retrieving a handle to an **AnalysisInstance** object that implements the BSA analysis tool;
- providing the **AnalysisInstance** with complete input information;
- invoking the **AnalysisInstance** to perform its function (via a synchronous or asynchronous mechanism);
- retrieving results generated by the BSA analysis tool execution; and
- when it no longer requires an **AnalysisInstance** (and its related input and output objects), invoking their removal from the system.

A Client can learn about processing events that occur during the execution of an **AnalysisInstance** either by asking the **AnalysisInstance** for its most recent processing event or listening to an event channel on which the **AnalysisInstance** publishes its events. A Client can also ask for an **AnalysisInstance**'s execution status.

2.2.1 General

```
//File: DsLSRAnalysis

#ifndef _DS_LSR_ANALYSIS_IDL_
#define _DS_LSR_ANALYSIS_IDL_

#pragma prefix "omg.org"

#include <orb.idl>
#include <CosPropertyService.idl>
#include <CosEventChannelAdmin.idl>
#include <CosLifeCycle.idl>
#include <TimeBase.idl>

module DsLSRAnalysis
{
    // ...
};

#endif // _DS_LSR_ANALYSIS_IDL_
```

#pragma prefix "omg.org"

To prevent name pollution and name clashing of IDL types, this module (and all modules defined in this specification) uses the pragma prefix that is the OMG's DNS name.

#include <orb.idl>

CORBA::TypeCode is used in **InputPropertySpec** and **OutputPropertySpec**.

#include <CosPropertyService.idl>

Properties are used in **AnalysisService** and **AnalysisInstance**.

#include <CosEventChannelAdmin.idl>

EventChannel is used in **AnalysisInstance**.

#include <CosLifeCycle.idl>

AnalysisInstance inherits from **LifeCycleObject**.

#include <TimeBase.idl>

TimeT is used in **TimeProgressEvent** and **JobControl**. **UtcT** is used in **JobControl**.

StringList

typedef sequence<string> StringList;	
Description:	Used to pass and return a set of **strings**.

2.2.2 AnalysisType

An **AnalysisType** provides information for a client to determine the types of BSA analyses available in the system. It can also be used to distinguish the type of analysis offered by an **AnalysisService**. An **AnalysisType** provides information sufficient to determine whether two **AnalysisServices** create identical BSA **AnalysisInstances**. Such information may be of use to a computation management subsystem such as a load balancing or queuing system. To provide enough information to distinguish analysis types, there are several attributes of an **AnalysisType**.

It is important to note that the **AnalysisType** is defined as a **valuetype** that can be extended by a vendor requiring additional attributes.

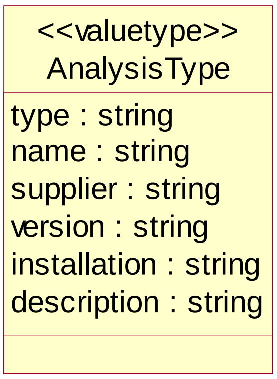


Figure 2-29 The AnalysisType valuetype

```
valuetype AnalysisType
{
    public string type;
    public string name;
    public string supplier;
    public string version;
    public string installation;
    public string description;
};
```

public string type;	
Description:	The **type** attribute is used to specify both the correct classification of the analysis as well as a qualifier to specify category and additionally, provides information about the inputs to the analysis. The classification of the analysis could come from the BSA specified classification hierarchy as well as it could come from a hierarchy defined by a certain installation. A '/' is used to delimit the qualifier and a '.' is used to delimit the general input kind. An example of a specified **type** attribute would be *alignment.collection/assembly.*
Return value:	Returns a **string**.

public string name;	
Description:	The **name** attribute is used to further identify the analysis in the system.
Return value:	Returns a **string**.

public string supplier;	
Description:	The **supplier** attribute is used to identify the supplier or vendor of a custom analysis implementation.
Return value:	Returns a **string**.

public string version;	
Description:	The **version** attribute specifies the particular form or variation of the analysis.
Return value:	Returns a **string**.

public string installation;	
Description:	The **installation** attribute is used to differentiate similar analysis implementations at a particular installation.
Return value:	Returns a **string**.

public string description;	
Description:	The **description** attribute is used to provide useful descriptive information about the **AnalysisInstances** created by the **AnalysisService**.
Return value:	Returns a **string**.

2.2.3 *InputPropertySpec*

An **InputPropertySpec** is used to provide metadata that describes required and optional input parameters used to perform an analysis. The **InputPropertySpec** provides an input name and **CORBA::TypeCode** to allow the client to interrogate the interface repository for more information about the analysis parameter. Additionally, there are some useful attributes that help the client determine if a parameter is optional or required, the default value of an input parameter if one exists, and finally some possible values useful for validation or user-interface presentation.

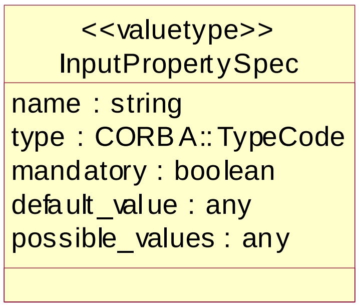


Figure 2-30 The InputPropertySpec valuetype

InputPropertySpec

```
valuetype InputPropertySpec
{
    public string               name;
    public CORBA::TypeCode      type;
    public boolean              mandatory;
    public any                  default_value;
    public any                  possible_values;
};
```

public string name;	
Description:	This is the name of the parameter that can be submitted to initialize the analysis.
Return value:	Returns a **string**.

public CORBA::TypeCode type;	
Description:	This is a **CORBA::TypeCode** allowing the client to find more detailed information in the interface repository about the data type.
Return value:	Returns a **CORBA::TypeCode**.

public boolean mandatory;	
Description:	The **mandatory** attribute specifies if the analysis requires the parameter with TRUE and if the parameter is optional with FALSE.
Return value:	Returns a **boolean**.

public any default_value;	
Description:	This attribute specifies the default value if one is applicable. If no default value is applicable, return a null in the **any**.
Return value:	Returns a CORBA **any**.

public any possible_values;	
Description:	This attribute specifies suggested allowed values that are applicable. If no possible values are applicable, return a null in the **any**.
Return value:	Returns a CORBA **any**.

InputPropertySpecList

typedef sequence<InputPropertySpec> InputPropertySpecList;	
Description:	Used to pass a set of **InputPropertySpecs**.

2.2.4 *OutputPropertySpec*

An **OutputPropertySpec** is used to provide metadata that describes each output value generated by an analysis. The **OutputPropertySpec** provides an output argument **name** and **CORBA::TypeCode** to allow the client to interrogate the interface repository for more information about the output value.

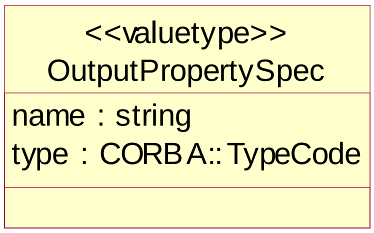


Figure 2-31 The OutputPropertySpec valuetype

OutputPropertySpec

```
valuetype OutputPropertySpec
{
	public string		name;
	public CORBA::TypeCode type;
};
```

public string name;	
Description:	This is the name of the identifier that contains an analysis output value.
Return value:	Returns a **string**.

public CORBA::TypeCode type;	
Description:	This is a **CORBA::TypeCode** allowing the client to find more detailed information in the interface repository about the data type.
Return value:	Returns a **CORBA::TypeCode**.

OutputPropertySpecList

typedef sequence<OutputPropertySpec> OutputPropertySpecList;	
Description:	Used to pass a set of **OutputPropertySpecs**.

2.2.5 AnalysisState

There are five defined analysis states:

1. **CREATED** - created but not yet invoked.
2. **RUNNING** – invoked.
3. **COMPLETED** – execution ended normally.
4. **TERMINATED_BY_REQUEST** – execution was terminated by a user request.
5. **TERMINATED_BY_ERROR** – execution terminated abnormally.

When an **AnalysisInstance** is first created it will be in the **CREATED** state. When the **AnalysisInstance** is successfully **run()** it will move into the **Running** state. In due course, the **AnalysisInstance** will then enter *either* the **COMPLETED**, **TERMINATED_BY_REQUEST** *or* **TERMINATED_BY_ERROR** state.

Note that an **AnalysisInstance** in the **TERMINATED_BY_REQUEST** or **TERMINATED_BY_ERROR** states may still have (partial, incomplete) results that can be retrieved by the client. There is no obligation that an implementation provides results in these two cases. Further, the results for an analysis that is in one of these two states is likely to be different than for an analysis that ran to normal completion. It is recommended that client software convey this information to the end-user.

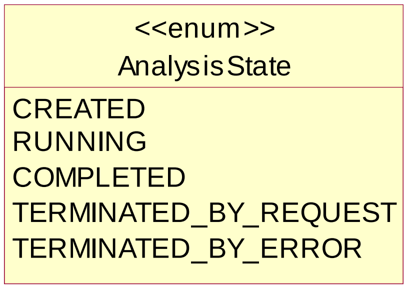


Figure 2-32 The AnalysisState enumeration

```
enum AnalysisState
{
      CREATED,                    // Instance has been created but not yet executed.
      RUNNING,                    // The analysis instance is running.
      COMPLETED,                  // The instance has completed execution.
      TERMINATED_BY_REQUEST,// The instance was terminated by user request.
      TERMINATED_BY_ERROR     // The instance terminated due to an error.
};
```

CREATED	**CREATED** should be used when the **AnalysisInstance** has been created but not yet invoked.
RUNNING	**RUNNING** should be used when the **AnalysisInstance** has been invoked.
COMPLETED	**COMPLETED** should be used to indicate that the execution of the **AnalysisInstance** ended normally.
TERMINATED_BY_REQUEST	**TERMINATED_BY_REQUEST** should be used to indicate that the execution of the **AnalysisInstance** was terminated by a user request.
TERMINATED_BY_ERROR	**TERMINATED_BY_ERROR** should be to indicate that the execution of the **AnalysisInstance** was terminated abnormally.

2.2.6 *AnalysisEvent*

There are five defined types of analysis events. They all inherit from the base valuetype, which has a single message string. For all events the string should give some free-form text description of the current progress.

- **StateChangedEvent**
- **HeartbeatProgressEvent**
- **PercentProgressEvent**
- **StepProgressEvent**
- **TimeProgressEvent**

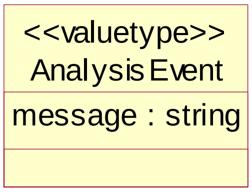


Figure 2-33 The AnalysisEvent valuetype

```
valuetype AnalysisEvent
{
     public string message;
};
```

public string message;	
Description:	For all events **message** should give some free-form text description of the current progress.
Return value:	Returns a **string**.

2.2.7 *Sub-types of AnalysisEvent*

If an analysis has a non-null event channel then it must publish **StateChangedEvents** onto that channel whenever the analysis enters a new state (apart from the **CREATED**) state.

The frequency of publication of other events onto the event channel is considered a quality of implementation issue. There is no restriction on the ordering of the events published onto the event channel.

An analysis may also publish other events (not necessarily derived from **AnalysisEvent**) onto the event channel. Clients, therefore, must be capable of dealing with unknown events (e.g. by discarding them).

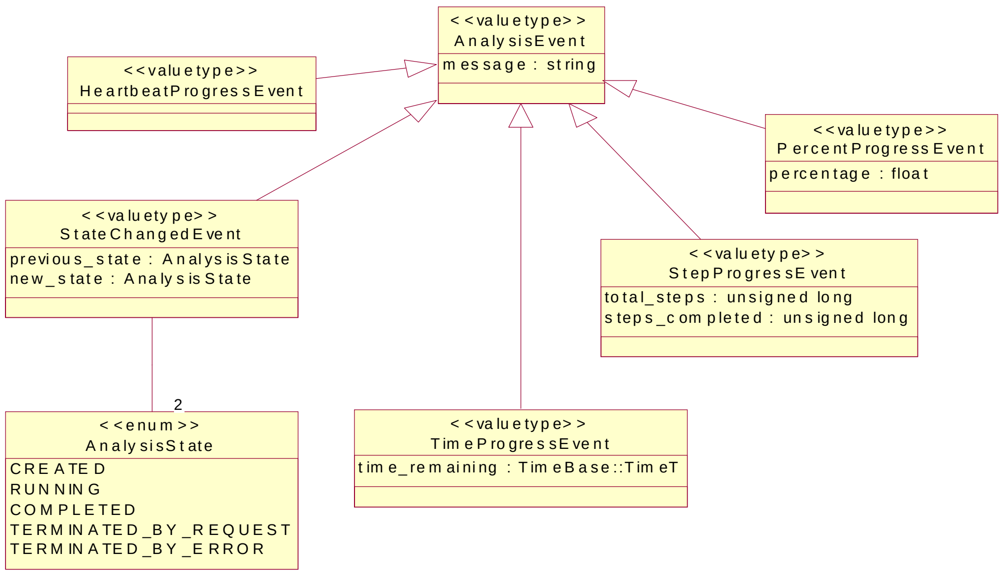


Figure 2-34 The Sub-types of AnalysisEvent

StateChangedEvent

StateChangedEvent indicates that an **AnalysisInstance** has changed from one of the five defined **AnalysisStates** to another. **StateChangedEvent** is truncatable to **AnalysisEvent**.

If an analysis has a non-null event channel then it must publish **StateChangedEvents** onto that channel whenever the analysis enters a new state (apart from the **CREATED**) state.

```
valuetype StateChangedEvent : truncatable AnalysisEvent
{
    public AnalysisState  previous_state;
    public AnalysisState  new_state;
};
```

public AnalysisState previous_state;	
Description:	Provides the previous state of the **AnalysisInstance**.
Return value:	Returns an **AnalysisState**.

public AnalysisState new_state;	
Description:	Provides the new state of the **AnalysisInstance**.
Return value:	Returns an **AnalysisState**.

HeartbeatProgressEvent

HeartbeatProgressEvent indicates that an **AnalysisInstance** is still alive and running. **HeartbeatProgressEvent** is truncatable to **AnalysisEvent**.

```
valuetype HeartbeatProgressEvent : truncatable AnalysisEvent
{
};
```

PercentProgressEvent

PercentProgressEvent provides information regarding the relative amount of work completed by an **AnalysisInstance** in terms of percentage complete. The percentage parameter must be greater or equal to 0 and less than or equal to 100. **PercentProgressEvent** is truncatable to **AnalysisEvent**.

```
valuetype PercentProgressEvent : truncatable AnalysisEvent
{
    public float percentage;
};
```

public float percentage;	
Description:	**percentage** must be greater or equal to 0 and less than or equal to 100.
Return value:	Returns a **float**.

TimeProgressEvent

TimeProgressEvent indicates the estimated completion time relative to the current time. There is no requirement that the estimated completion time decreases! **TimeProgressEvent** is truncatable to **AnalysisEvent**.

```
valuetype TimeProgressEvent : truncatable AnalysisEvent
{
    public TimeBase::TimeT time_remaining;
};
```

public TimeBase::TimeT time_remaining;	
Description:	Indicates the estimated completion time relative to the current time.
Return value:	Returns a **TimeBase::TimeT**.

StepProgressEvent

StepProgressEvent indicates the total number of steps to be executed by an **AnalysisInstance** and the number of steps completed so far. Multiple **StepProgressEvents** with the same progress string must have the same total number of steps. The **steps_completed** parameter must be less than or equal to the **total_steps** parameter. **StepProgressEvent** is truncatable to **AnalysisEvent**.

```
valuetype StepProgressEvent : truncatable AnalysisEvent
{
    public unsigned long total_steps;
    public unsigned long steps_completed;
};
```

public unsigned long total_steps;	
Description:	Indicates the total number of steps to be executed by the **AnalysisInstance**. The **steps_completed** parameter must be less than or equal to the **total_steps** parameter.
Return value:	Returns an **unsigned long**.

public unsigned long steps_completed;	
Description:	Indicates the number of steps completed so far. The **steps_completed** parameter must be less than or equal to the **total_steps** parameter.
Return value:	Returns an **unsigned long**.

2.2.8 *AnalysisService*

An **AnalysisService** is a logical representation of a particular type of a BSA analysis tool available within a system. An **AnalysisService** provides enough information to distinguish the service it provides from those offered by other **AnalysisServices**.

An **AnalysisService** provides metadata that describes input to its **AnalysisInstances** and the output generated by its **AnalysisInstances**. Metadata describing input and output parameters is available to the client in either IDL valuetypes or both IDL valuetypes and XML strings. If both are used, the information available in the IDL structures and XML strings must not be contradictory. Obviously there is some information, such as constraints expressed in OCL (Object Constraint Language), that will only be available in the XML strings. Metadata is required for a compliant implementation.

An **AnalysisService** creates and returns references to **AnalysisInstance** objects that implement the BSA analysis tool it represents. Arguments to create an **AnalysisInstance** are in the form of **CosPropertyService::Properties**. Before returning an **AnalysisInstance**, the input arguments must be checked for correctness (according to the criteria represented in the metadata describing the **AnalysisService's** input parameters).

The client that receives the returned reference to an **AnalysisInstance** is responsible for the lifecycle management of that instance along with the objects populating the **AnalysisInstance's** input parameters and output parameters.

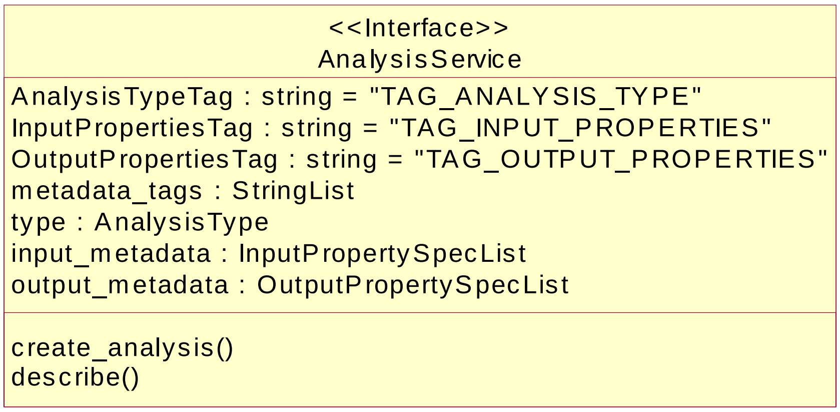

<<Interface>>
AnalysisService
AnalysisTypeTag : string = "TAG_ANALYSIS_TYPE"
InputPropertiesTag : string = "TAG_INPUT_PROPERTIES"
OutputPropertiesTag : string = "TAG_OUTPUT_PROPERTIES"
metadata_tags : StringList
type : AnalysisType
input_metadata : InputPropertySpecList
output_metadata : OutputPropertySpecList
create_analysis()
describe()

Figure 2-35 The AnalysisService interface

MetaData

typedef string MetaData;	
Description:	Used to pass and return a **string** containing XML metadata.

DoesNotExistException

exception DoesNotExistException { };	
Description:	The DoesNotExistException exception is raised if the **tagname** used in **describe**() does not exist in the metadata.

AnalysisService

```
interface AnalysisService
{
    const string AnalysisTypeTag         = "TAG_ANALYSIS_TYPE";
    const string InputPropertiesTag      = "TAG_INPUT_PROPERTIES";
    const string OutputPropertiesTag     = "TAG_OUTPUT_PROPERTIES";

    readonly attribute StringList metadata_tags;
    MetaData describe(in string tagname)
        raises (DoesNotExistException);

    readonly attribute AnalysisType type;
    readonly attribute InputPropertySpecList input_metadata;
    readonly attribute OutputPropertySpecList output_metadata;

    AnalysisInstance create_analysis (in CosPropertyService::Properties input)
        raises (CosPropertyService::MultipleExceptions);
};
```

const string AnalysisTypeTag = "TAG_ANALYSIS_TYPE";
const string InputPropertiesTag = "TAG_INPUT_PROPERTIES";
const string OutputPropertiesTag = "TAG_OUTPUT_PROPERTIES";	

Description:	The **AnalysisService** interface defines a set of **const strings** that indicates the types of required metadata. The strings correspond to the three attributes described below.

readonly attribute StringList metadata_tags;	
Description:	Provides the set of metadata tags for this analysis. The list must include the three **const strings** listed above.
Return value:	Returns a **StringList**.

MetaData describe(in string tagname) **raises (DoesNotExistException);**	
Description:	**describe()** returns an XML string containing the metadata corresponding to the tagname parameter. If metadata is available as XML, **describe()** must support all tagnames returned by the metadata_tags attribute.
Return value:	Returns a **MetaData** string containing XML.
Exceptions:	Raises DoesNotExistException if the tagname parameter is not one of the list returned by the metadata_tags attribute. Raises CORBA::NO_IMPLEMENT with standard minor code 6 if metadata is not available as XML.

readonly attribute AnalysisType type;	
Description:	**type()** returns the **AnalysisType** structure. This structure must be populated.
Return value:	Returns an **AnalysisType**.

readonly attribute InputPropertySpecList input_metadata;	
Description:	**input_metadata()** returns information about input parameters in IDL structure form. This structure must be populated.
Return value:	Returns an array of **InputPropertySpecs**.

readonly attribute OutputPropertySpecList output_metadata;	
Description:	**output_metadata()** returns information about output parameters in IDL structure form. This structure must be populated.
Return value:	Returns an array of **OutputPropertySpecs**.

AnalysisInstance create_analysis (in CosPropertyService::Properties input) raises (CosPropertyService::MultipleExceptions);	
Description:	Arguments to create an **AnalysisInstance** are in the form of **CosPropertyService::Properties**. Before returning an **AnalysisInstance**, the input arguments must be checked for correctness (according to the criteria represented in the metadata describing the **AnalysisService**'s input parameters).
Return value:	Returns an **AnalysisInstance**.
Exceptions:	Raises CosPropertyService::MultipleExceptions if the input parameters are incorrect for this analysis. The metadata should be consulted for information about the input parameters needed by this analysis.

2.2.9 JobControl

Along with its basic interface, an **AnalysisInstance** implements a **JobControl** interface. Via the **JobControl**, clients invoke and terminate **AnalysisInstance** execution and retrieve execution information (e.g., execution duration, creation time).

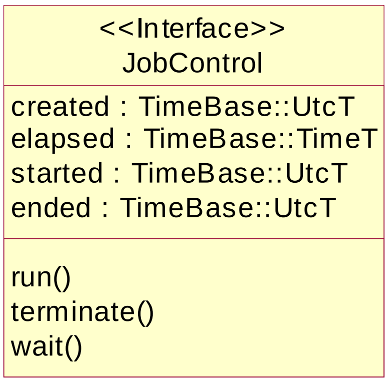


Figure 2-36 The JobControl interface

NotRunnable

exception NotRunnable { };	
Description:	The NotRunnable exception is raised if the analysis cannot be run (e.g., the service is currently unavailable). Raised by **run()**. This exception should not be used to indicate incorrect inputs. **CosPropertyService::MultipleExceptions** should be used instead.

NotRunning

exception NotRunning { };	
Description:	The NotRunning exception is raised if the analysis is not running. Raised by **terminate()**.

NotTerminated

```
exception NotTerminated
{
    string reason;
};
```

Description:	The NotTerminated exception is raised if the analysis is not terminated. Raised by **terminate()**.
Return value:	Returns a **string** containing the reason the analysis could not be terminated.

JobControl

```
interface JobControl
{
    readonly attribute TimeBase::UtcT created;
    readonly attribute TimeBase::TimeT elapsed;
    readonly attribute TimeBase::UtcT started;
    readonly attribute TimeBase::UtcT ended;

    void run()
        raises (NotRunnable, CosPropertyService::MultipleExceptions);
    void terminate()
        raises (NotRunning, NotRunnable, NotTerminated);
    void wait()
        raises (NotRunning, NotRunnable);
};
```

readonly attribute TimeBase::UtcT created;	
Description:	Indicates the time the **AnalysisInstance** was created.
Return value:	Returns a **TimeBase::UtcT**.

readonly attribute TimeBase::TimeT elapsed;	
Description:	Indicates the elapsed time since the analysis was started using **run()**.
Return value:	Returns a **TimeBase::TimeT**.

readonly attribute TimeBase::UtcT started;	
Description:	Indicates the time the analysis was started.
Return value:	Returns a **TimeBase::UtcT**.

readonly attribute TimeBase::UtcT ended;	
Description:	Indicates the time the analysis ended.
Return value:	Returns a **TimeBase::UtcT**.

void run() **raises (NotRunnable, CosPropertyService::MultipleExceptions);**	
Description:	The **run()** method invokes the **AnalysisInstance** to run asynchronously
Exceptions:	Raises NotRunnable if the analysis cannot be run (e.g., the service is currently unavailable). Raises CosPropertyService::MultipleExceptions if the inputs are not correct.

void terminate() **raises (NotRunning, NotRunnable, NotTerminated);**	
Description:	**terminate()** ends a currently running analysis.
Exceptions:	Raises NotRunning if the analysis is not running. Raises NotRunnable if the analysis cannot be run (e.g., the has already completed or terminated). Raises NotTerminated if the analysis was not terminated.

void wait() **raises (NotRunning, NotRunnable);**	
Description:	The **wait()** method blocks the client until service execution completes.
Exceptions:	Raises NotRunning if the analysis is not running. Raises NotRunnable if the analysis cannot be run (e.g., the has been terminated).

The following table summarizes the behavior of the three methods for each possible state of the **AnalysisInstance**.

Table 2-7 JobControl State Transition Table

State of AnalysisInstance	**run()**	**wait()**	**terminate()**
CREATED	start it and change status to **RUNNING**	raise **NotRunning**	raise **NotRunning**

Table 2-7 JobControl State Transition Table

State of AnalysisInstance	run()	wait()	terminate()
RUNNING	raise **NotRunnable**	block until finished	kill it and change status to **TERMINATED_BY_ REQUEST** or raise **NotTerminated**
COMPLETED	raise **NotRunnable**	do nothing	raise **NotRunnable**
TERMINATED_BY_REQUEST	raise **NotRunnable**	raise **NotRunnable**	raise **NotRunnable**
TERMINATED_BY_ERROR	raise **NotRunnable**	raise **NotRunnable**	raise **NotRunnable**

2.2.10 AnalysisInstance

An **AnalysisInstance** object is responsible for invoking an underlying BSA analysis tool.

An **AnalysisInstance** can be used in either a synchronous or an asynchronous mode to support clients with various needs. The **run()** method invokes the **AnalysisInstance** to run asynchronously. If the client wants to be blocked waiting for the underlying BSA analysis tool to run to completion, it will invoke the **run()** method, followed immediately by the **wait()** method, which will block the client until service execution completes.

An **AnalysisInstance** must ensure it can be executed only once, ensuring a unique coupling of inputs and results. If a client wants to employ an **AnalysisInstance** identical to one it has already invoked, the client must create a new **AnalysisInstance**, via an **AnalysisService**, and invoke it as a separate instance.

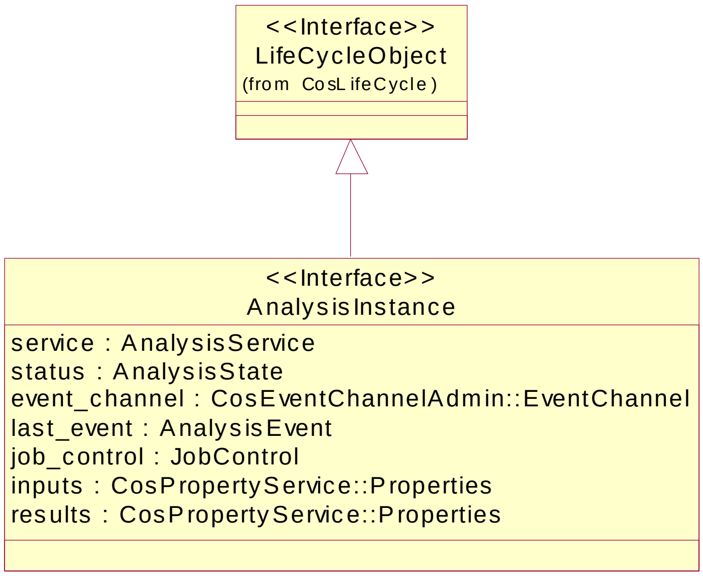
An **AnalysisInstance** makes available two kinds of execution information: execution status and analysis events.

- An **AnalysisInstance** object must offer:
  - the **AnalysisService** that created this **AnalysisInstance**;
  - its execution status (one of the enumerated **AnalysisState** values);
  - the **EventChannel**, which may be null, to which it publishes its analysis events;
  - the last event that occurred;
  - the **JobControl** that clients use to control the execution;
  - the input **Properties** used in creation of this **AnalysisInstance**;
  - an output **Properties** containing the results generated by the execution of the underlying BSA analysis.

An **AnalysisInstance** is responsible for ensuring that the results of the BSA analysis tool it represents are populated properly in its **results**.

To retrieve the results generated by an **AnalysisInstance**, clients use the **get_result()** method. It takes a list of strings (the strings representing named members of the **OutputPropertySpecList**) as an argument. If the BSA analysis tool underlying the **AnalysisInstance** terminated before it completed, either due to a client request or an execution failure, some “partial” results may be available to the client in the **results**.

As in all CORBA systems, an implementation of this system may choose to enforce a policy regarding automatically removing CORBA objects, such as **AnalysisInstances** that appear to have been abandoned by clients.



<<Interface>>
LifeCycleObject
(from CosLifeCycle)
<<Interface>>
AnalysisInstance
service : AnalysisService
status : AnalysisState
event_channel : CosEventChannelAdmin::EventChannel
last_event : AnalysisEvent
job_control : JobControl
inputs : CosPropertyService::Properties
results : CosPropertyService::Properties

Figure 2-37 The AnalysisInstance interface

```
interface AnalysisInstance : CosLifeCycle::LifeCycleObject
{
    readonly attribute AnalysisService service;
    readonly attribute AnalysisState   status;
    readonly attribute CosEventChannelAdmin::EventChannel event_channel;
    readonly attribute AnalysisEvent                      last_event;
    readonly attribute JobControl                         job_control;
    readonly attribute CosPropertyService::Properties     inputs;
    readonly attribute CosPropertyService::Properties     results;
    CosPropertyService::Properties get_result(in StringList name_list);
};
```

readonly attribute AnalysisService service;	
Description:	Refers to the **AnalysisService** that created this **AnalysisInstance**.
Return value:	Returns an **AnalysisService**.
Exceptions:	Raises CORBA::OBJECT_NOT_EXIST with standard minor code 3 if the **AnalysisService** is no longer available.

readonly attribute AnalysisState status;	
Description:	Provides the current status of the analysis.
Return value:	Returns one of the enumerated **AnalysisState** values. The values are **CREATED**, **RUNNING**, **COMPLETED**, **TERMINATED_BY_REQUEST**, and **TERMINATED_BY_ERROR**.

readonly attribute CosEventChannelAdmin::EventChannel event_channel;	
Description:	Provides the **EventChannel** to which the **AnalysisInstance** publishes its analysis events.
Return value:	Returns a **CosEventChannelAdmin::EventChannel**.

readonly attribute AnalysisEvent last_event;	
Description:	Provides the last event that occurred during execution. If the **AnalysisInstance** is still in the **CREATED** state, the **last_event** should be a plain **AnalysisEvent**.
Return value:	Returns an **AnalysisEvent**.

readonly attribute JobControl job_control;	
Description:	Provides the management interface that clients can use to control the execution of the analysis.
Return value:	Returns a **JobControl**.

readonly attribute CosPropertyService::Properties inputs;	
Description:	Provides the input **Properties** that were used in this **AnalysisInstance**'s execution.
Return value:	Returns a **CosPropertyService::Properties**.

readonly attribute CosPropertyService::Properties results;	
Description:	Provides the output **Properties** containing the results generated by the execution of the underlying BSA analysis. Note 1: An **AnalysisInstance** in the **RUNNING** or **TERMINATED_BY_REQUEST** or **TERMINATED_BY_ERROR** states may still have (partial, incomplete) results that can be retrieved by the client. There is no obligation that an implementation provides results in these three cases. Further, the results for an analysis that is in one of these three states is likely to be different than for an analysis that ran to normal completion. It is recommended that client software convey this information to the end-user. Note 2: An **AnalysisInstance** in the **CREATED** state returns an empty list of results.
Return value:	Returns a **CosPropertyService::Properties**.

CosPropertyService::Properties get_result(in StringList name_list);	
Description:	The **get_result()** method takes a list of strings (the strings representing named members of the **OutputPropertySpecList**) as an argument and returns the associated results. Note 1: An **AnalysisInstance** in the **RUNNING** or **TERMINATED_BY_REQUEST** or **TERMINATED_BY_ERROR** states may still have (partial, incomplete) results that can be retrieved by the client. There is no obligation that an implementation provides results in these three cases. Further, the results for an analysis that is in one of these three states is likely to be different than for an analysis that ran to normal completion. It is recommended that client software convey this information to the end-user. Note 2: An **AnalysisInstance** in the **CREATED** state returns an empty list of results.
Return value:	Returns a **CosPropertyService::Properties**.

2.2.11 Sequence Diagrams

The following sequence diagrams show how the analysis machinery is used. The diagrams are examples of the steps necessary for both synchronous and asynchronous invocation of an analysis service and retrieving its results.

Synchronous invocation can be achieved without using any **EventChannel** interface. The client is blocked in **wait()** method until the analysis is finished.

Asynchronous invocation, using an **EventChannel**, can follow a "callback" pattern where the server regularly pushes events back to an object prepared by the client, or the client can repeatedly poll the server.

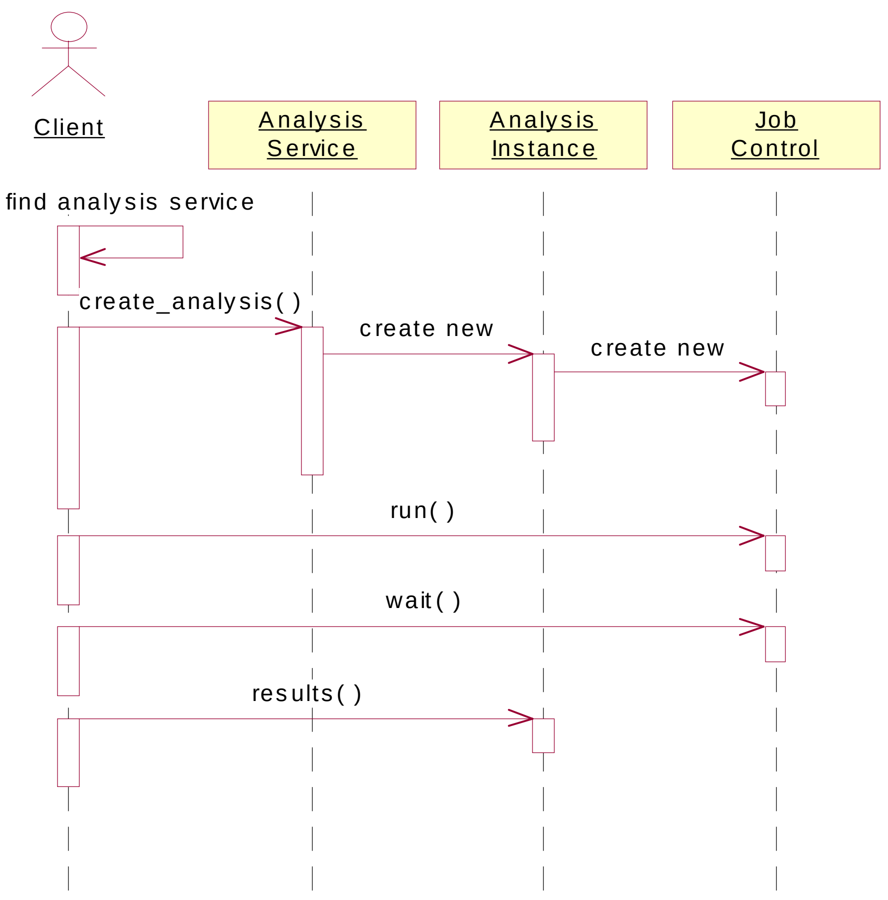


Figure 2-38 Synchronous invocation without using an EventChannel

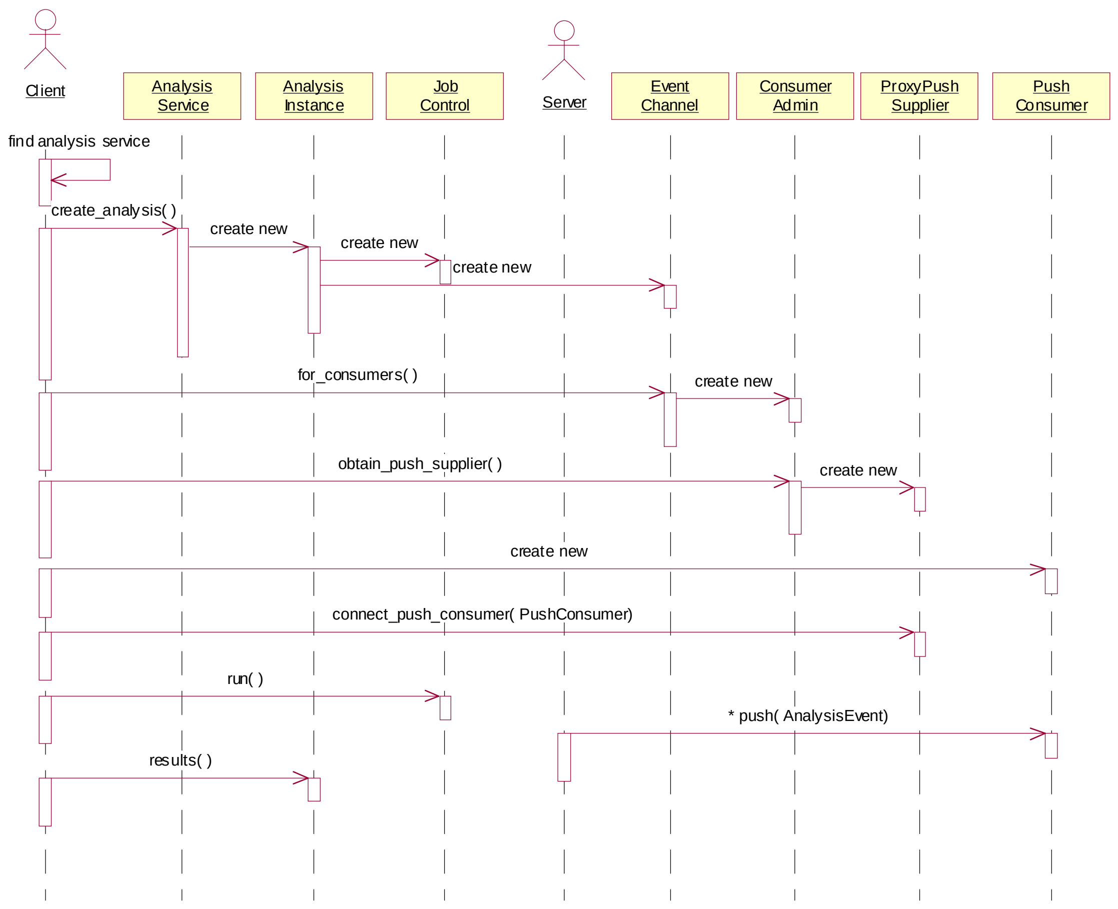


Figure 2-39 Asynchronous invocation, using an EventChannel and callbacks

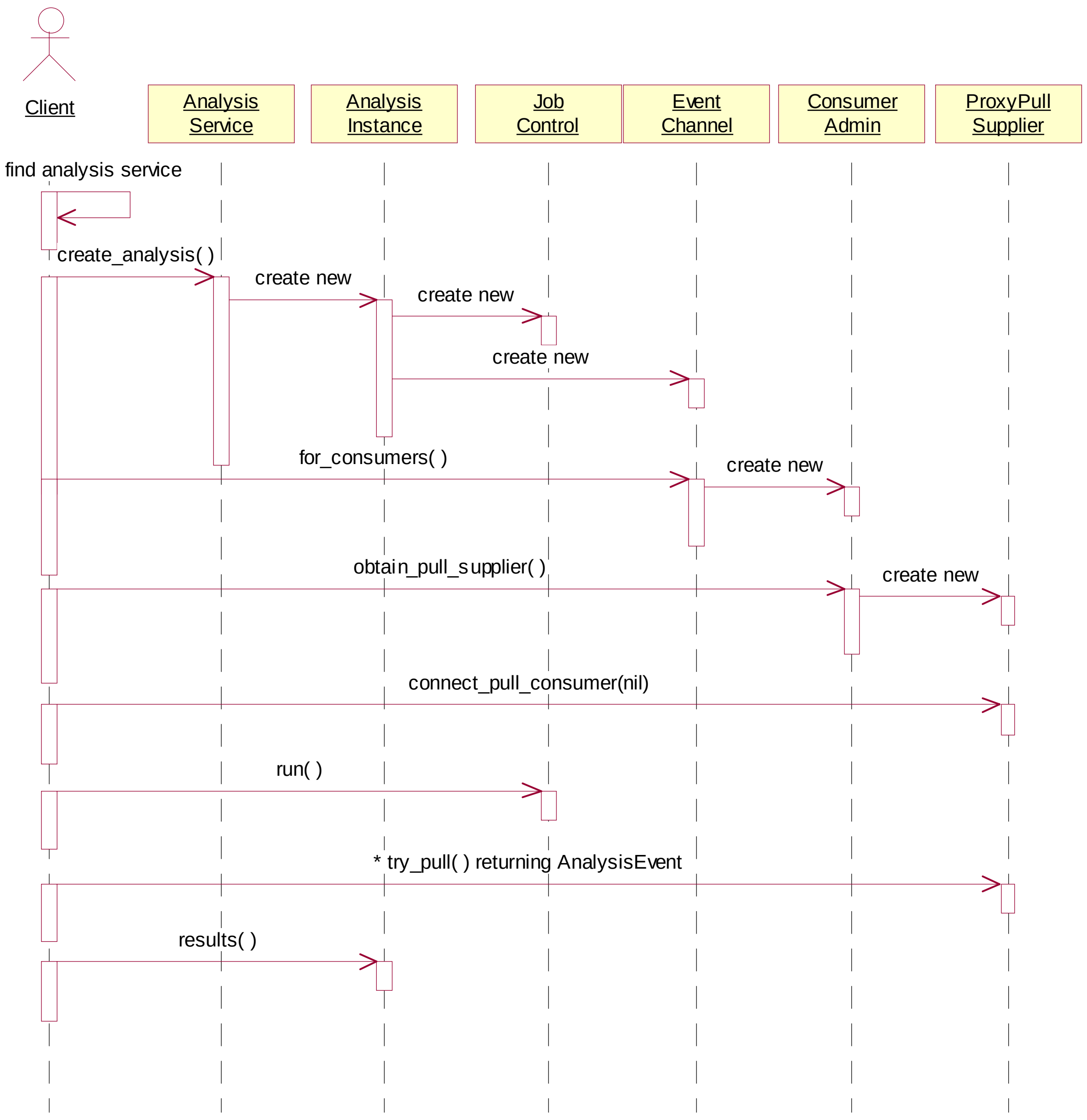


Figure 2-40 Asynchronous invocation, using an EventChannel and polling

Domain Model 3

Contents

This chapter contains the following sections.

The domain model is expressed in XML. A simple classification of analyses follows the explanation of XML metadata. The DTD and the entire XML file can be found in Appendix D.

3.1 XML Metadata

Metadata is used in the **AnalysisService** for analysis type, input and output objects to represent object implementation detail that has been abstracted out of the interface in favor of using the standard, common BSA analysis interface. This can provide semantic information beyond that provided by the IDL syntax, although information provided through XML must not be contradictory with information available through IDL structures. XML has been chosen as the language with which to represent the object metadata. This section discusses the strategy for using the XML based metadata representation.

3.1.1 Role of XML

"Standard Generalized Markup Language (SGML), which became an ISO 8879 standard in 1986, was the result of a decade's long effort to produce a language for writing human consumable text that at the same time is machine processable. Hypertext Markup Language (HTML), a limited subset of SGML, is one of the driving forces behind the success of the internet. HTML is non-extensible and primarily designed to support rendering in a browser and a limited amount of user interaction. Extensible Markup Language (XML) is a larger subset of SGML which overcomes the non-extensible nature of HTML and reintroduces support for the machine processing of text via the definition of user specified tag sets. Since its inception, XML has offered the prospect of overcoming the limitations of HTML without unduly burdening development of processing software as has been the case with SGML based systems. Unlike HTML, XML does not rely on a fixed set of tags. Arbitrary tag sets can be defined via use of a DTD. However, XML eliminates several features of SGML which make it difficult to parse and therefore difficult to process SGML documents. In particular, begin and end tags are both required and serve to reduce ambiguity in the processing of the hierarchical structure of XML documents, relative to SGML documents. In short, XML provides a standardized, non-proprietary capability to represent arbitrary structural information in a way that supports development of parsers and other types of processing of that structural information. Thus, XML opens up the possibility of automated processing and interchange of information stored in the form of XML documents.

With respect to metadata, it opens up the possibility of accessing metadata at runtime and using the structural information provided by the XML based tags to process and transform that metadata. For example, the metadata for two separate processes could be used at runtime to connect the output of one process to the input of another process via conversion of the output format of the first process into the input format of the second process."

[Concept Five Technologies, Inc., *Trident Next Generation Metadata Design and Generation Manual version 1.01*, pages 3-4, Copyright © 1998, 1999 by Hitachi, Ltd. and Concept Five Technologies, Inc.]

3.1.2 Role of DTD

As the XML proposal most succinctly puts, "The XML document type declaration contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or DTD."

The markup grammar is a generic set of keywords, naming syntax, occurrence and connector terms prescribed in the XML standard that the document structure designer wishes to use to express literally any real world semantic notion. The basic markup keywords are ELEMENT, ENTITY and ATTRIBUTE although there are dozens of others to round out the language. Any set of key words could have been chosen. Microsoft Word has its set of formatting keywords and arguments that allow a .doc file to carry a formidable amount of information around for future processing. WordPerfect used to allow a document writer to make these codes visible and directly editable at the click of a menu item. And of course, there is post script.

There are many other document code sets, all of which are proprietary. Document processing code that operates on these proprietarily marked up documents must necessarily also be proprietary. Enter the DTD, or Document Type Definition. ISO 8879 makes standard these markup codes so non-proprietary document software can be developed.

Hypertext Markup Language (HTML) is an example of a markup language. Although HTML is not based on a DTD it does adhere to a standard and stems from SGML. HTML was designed so that processing code could be developed for rendering HTML based documents in a browser. The HTML standard (currently 4.0) specifies the structure of valid HTML documents. Changing one of the tags in this standard from <H3> to <J3> has the potential to break all the processing code that relies on the use of the standard, which is why changes to HTML are only made infrequently. Recently, DTD's have been developed for HTML, but these DTD's do not adhere strictly to the standard, and are not widely used.

In general, DTD's make it possible to specify the grammars of various domains so that companies creating XML documents in these domains can interact with each other. For example, there is a DTD for the representation of chemical formulas in XML. Companies complying with the grammar for this domain can expect to be able to exchange XML documents describing chemical formulas and be able to use any processing code designed to operate in this domain. For example, processing code that accepts XML based descriptions of chemical formulas and creates graphical representations of the formulas should be able to handle any documents complying with the DTD.

At the present time, DTD's are being generated for many different domains. The Dublin Core is a DTD which provides a tag set designed for use in the description of Internet information resources and which is patterned after the information in a card catalog. The UML DTD which is derived from the XMI specification covers the domain of object modeling and is based on the UML semantics document. This DTD is likely to become the standard for the description of object models in XML. Companies which produce documents which comply with standardized DTD's will be able to exploit any processing developed for use with those standardized DTD's."

[Concept Five Technologies, Inc., *Trident Next Generation Metadata Design and Generation Manual version 1.01*, pages 3-4, Copyright © 1998, 1999 by Hitachi, Ltd. and Concept Five Technologies, Inc.]

3.1.3 Domain Metadata

Interoperability requires convergence on data semantics description capabilities. The metadata in a BSA environment includes a description of the CORBA interfaces supported as well as the meta semantics related to specification of the analysis and input and output types supported by a particular analysis interface. The BSA metadata for the analysis type, inputs and outputs allows for the support of well understood multiple execution paths supported through the same simple interface.

The metadata provided by the valuetypes and XML is required to facilitate interoperability for analyses, inputs and outputs. Interoperability is achieved by providing run-time information about parameters required to perform an analysis. The client can dynamically interrogate the analysis service, learn about the input parameters, populate the input property set and perform the analysis. When the analysis is finished, the client can dynamically check the analysis service to learn about the output properties. The client can use this knowledge to dissect the outputs into information of interest.

The elements in the DsLSRAnalysis DTD correspond to the attributes in the previously defined **AnalysisType**, **InputPropertySpec**, and **OutputPropertySpec** valuetypes. In addition to the required valuetypes, the XML metadata may be available for the implementation to provide data about the analyses.

The attributes "type" used in the elements "input" and "output" contain a string representation of the **CORBA::TypeCode**. These representations conform to the following rules.

1. Basic IDL types are represented by a string containing the name of the type. The type is derived from the **CORBA TypeCode's TCKind** by deleting the leading "tk_". This rule follows the convention used in section 5.3.10.2 (CorbaTypeName) of the XMI 1.0 specification (formal/00-06-01).

   Example: the string representation of the type **long** is "ong;" that of **unsigned long long** is "ulonglong."

2. Sequences of basic IDL types are represented by a string containing the type-specifier in IDL syntax without any spaces. That is, a sequence of **XXXs** is coded as "sequence<XXX>" where XXX is the name of the string found using rule 1.

   Example: A sequence of **longs** is represented by "“equence<long>."

3. For other data types, the repository ID is used.

   Example: the **BioSequence** is represented by "IDL:omg.org/DsLSRBioObjects/BioSequence:1.1."

The elements have the same definition as the valuetype attributes previously specified. It is important to highlight the analysis type format. Again, the *type* element is used to specify both the correct classification of the analysis as well as a qualifier to specify category and additionally, provides information about the inputs to the analysis. The classification of the analysis could come from the BSA specified classification hierarchy as well as it could come from a hierarchy defined by a certain installation. A '/' is used to delimit the qualifier and a '.' is used to delimit the general input kind. An example of a specified *type* element would be *alignment.collection/assembly.*

The DTD has three places where vendor extension is available. The analysis, input and output elements specify an extension element that can be any valid content.

The following text presents the DTD for Biomolecular Sequence Analysis.

```
<!ELEMENT DsLSRAnalysis (analysis)+>

<!ELEMENT analysis (description?, input*, output*, analysis_extension?)>

<!ATTLIST analysis
    type            CDATA #REQUIRED
    name            CDATA #IMPLIED
    version         CDATA #IMPLIED
    supplier        CDATA #IMPLIED
    installation    CDATA #IMPLIED>

<!ELEMENT description ANY>
<!ELEMENT extension ANY>

<!ELEMENT input (default?, allowed*, input_extension?)>

<!ATTLIST input
```

```
    type            CDATA #REQUIRED
    name            CDATA #REQUIRED
    mandatory       (true|false) "false">

<!ELEMENT default (#PCDATA)>
<!ELEMENT allowed (#PCDATA)>

<!ELEMENT output (output_extension?)>

<!ATTLIST output
    type            CDATA #REQUIRED
    name            CDATA #REQUIRED>

<!ENTITY % vendor_analysis_tags "<!ELEMENT analysis_extension
EMPTY>">
<!ENTITY % vendor_input_tags    "<!ELEMENT input_extension EMPTY>">
<!ENTITY % vendor_output_tags   "<!ELEMENT output_extension
EMPTY>">

%vendor_analysis_tags;
%vendor_input_tags;
%vendor_output_tags;
```

The following text provides example XML that would be used with respect to the DsLSRAnalysis DTD.

```
<?xml version="1.0" ?>
<!DOCTYPE DsLSRAnalysis SYSTEM "DsLSRAnalysis.dtd" >

<DsLSRAnalysis>
    <ANALYSIS TYPE = "search.list">
        <INPUT
            NAME = "query_sequence"
            TYPE = "IDL:omg.org/DsLSRBioObjects/BioSequence:1.0"
            MANDATORY = "true">
        </INPUT>
        <INPUT
            NAME = "sequence_list"
            TYPE = "IDL:omg.org/DsLSRBioObjects/BioSequenceList:1.0"
            MANDATORY = "true">
        </INPUT>
        <OUTPUT
            NAME = "search_result"
            TYPE = "IDL:omg.org/DsLSRBioObjects/SearchResult:1.0">
        </OUTPUT>
    </ANALYSIS>
</DsLSRAnalysis>
```

3.2 Classification of Analyses

This classification of analyses consists of three broad categories: searching, alignment, and utilities. Commonly used analyses are nicely partitioned into these categories.

3.2.1 Searching

Searching includes the broad category of similarity searching analyses. BLAST, FastA, and Smith-Waterman fall into this group. Searching can include querying **BioSequences** to identify **Annotations** that meet specified criteria. Searching also includes finding patterns and motifs in **BioSequences**. The results of these searches are **SeqRegions**. Examples include analyses such as PROSITE, BLOCKS, PRINTS, as well as most gene and ORF finding algorithms (e.g., GRAIL, GeneScan, GeneFind, and GLIMMER). It also includes identifying potential restriction enzyme and proteolytic cleavage sites.

The result of a search is a **SearchResult**. A **SearchResult** contains an array of **SearchHits**, which may be the specialized **SimilaritySearchHits**.

The searching hierarchy is:

- search (against a list, collection, or database)
- search/annotation
- search/region
- search/similarity (against a list, collection, or database)

3.2.2 Alignment

The alignment category includes both pairwise and multiple alignments. No distinction is made. The result of either is an **Alignment**.

A sequence assembly contains both aligned sequences and unaligned sequences (fragments). The aligned sequences are represented by an **Alignment**. If one considers a phylogeny as an alignment of alignments, it too falls in this category.

The alignment hierarchy is:

- alignment (of a list or collection)
- alignment/assembly (of a list or collection)
- alignment/phylogeny

3.2.3 Utilities

There are several simple analyses that could either be viewed as analyses or simply provided as methods on an appropriately typed **BioSequence**. We decided to view them as simple analysis. This allowed us to keep the **BioSequence** interface simple. For example, simple sequence translation, using the standard genetic code, is provided by **NucleotideSequence's** methods **translate_seq()** and **translate_seq_region()**. A more sophisticated sequence translation, allowing a user specified **GeneticCode**, is provided here.

The utilities category provides:

- utility/molecular_weight
- utility/residue_composition
- utility/ambiguous_residues
- utility/gc_content
- utility/isoelectric_point
- utility/translate_seq (uses GeneticCode)
- utility/translate_seq.seq_region (uses GeneticCode)

References A

A.1 List of References

Object Management Group. 1998. Biomolecular Sequence Analysis RFP. OMG Document lifesci/98-03-05.

Object Management Group. 1998. The Common Object Request Broker: Architecture and Specification, v2.2. OMG Document formal/98-07-01.

Object Management Group. 1998. CORBAservices: Common Object Services Specification. OMG Document formal/98-12-09.

Object Management Group. 1998. CORBAservices: Common Object Services IDL. OMG Document formal/98-10-53.

Object Management Group. 1998. CORBA v2.3a - Core final revision. OMG PC Document ptc/98-12-04.

Object Management Group. 1998. Interoperable Naming Service. OMG Document orbos/98-10-11.

Object Management Group. 1998. Joint Revised Objects by Value Submission - with Errata. OMG TC Document orbos/98-01-18.

Object Management Group. 1998. OMG IDL Style Guide. OMG Document ab/98-06-03.

Object Management Group. 2000. OMG XML Metadata Interchange (XMI) Specification. OMG Document formal/00-06-01.

Bairoch, Amos, et al. 1997. The Swiss-Prot Protein Sequence Data Bank User Manual. Release 35; November 1997.

Baldi, Pierre and Søren Brunak. 1998. Bioinformatics: The Machine Learning Approach. The MIT Press. ISBN: 0-262-02442-X.

Baxevanis, Andreas D. and B.F. Francis Ouellette, eds. 1998. Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins. Wiley-Interscience. ISBN: 0-471-19196-5.

Elzanowski, Andrzej (Anjay) and Jim Ostell, compilers. 1996. The Genetic Codes. National Center for Biotechnology Information (NCBI). http://www.ncbi.nlm.nih.gov/htbin-post/Taxonomy/wprintgc?mode=t.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. ISBN: 0-201-63361-2.

Gusfield, Dan. 1997. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge Univ Pr (Short). ISBN: 0-521-58519-8.

IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) symbols for amino acids. http://www.chem.qmw.ac.uk/iupac/AminoAcid/.

IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) symbols for nucleic acids. http://www.chem.qmw.ac.uk/iubmb/misc/naseq.html.

IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) biostandards. http://www.chem.qmw.ac.uk/iubmb/nomenclature/.

Lander, Eric S., and Michael S. Waterman, eds. 1995. Calculating the Secrets of Life: Applications of the Mathematical Sciences in Molecular Biology. National Academy Press. ISBN: 0-309-04886-9.

National Center for Biotechnology Information, et al. 1997. The DDJB/EMBL/GenBank Feature Table: Definitions. Version 2.0. December 15, 1997.

Waterman, Michael S. 1995. Introduction to Computational Biology: Maps, Sequences, and Genomes. Chapman & Hall. ISBN: 0-412-99391-0.

Genetic Codes B

The genetic codes listed below were compiled by Andrzej (Anjay) Elzanowski and Jim Ostell (National Center for Biotechnology Information). See http://www.ncbi.nlm.nih.gov/htbin-post/Taxonomy/wprintgc?mode=t. “i” indicates initiation and alternative initiation codons.

B.1 Standard

TTT	F Phe	TCT	S Ser	TAT	Y Tyr	TGT	C Cys
TTC	F Phe	TCC	S Ser	TAC	Y Tyr	TGC	C Cys
TTA	L Leu	TCA	S Ser	TAA	* Ter	TGA	* Ter
TTG	L Leu i	TCG	S Ser	TAG	* Ter	TGG	W Trp
CTT	L Leu	CCT	P Pro	CAT	H His	CGT	R Arg
CTC	L Leu	CCC	P Pro	CAC	H His	CGC	R Arg
CTA	L Leu	CCA	P Pro	CAA	Q Gln	CGA	R Arg
CTG	L Leu i	CCG	P Pro	CAG	Q Gln	CGG	R Arg
ATT	I Ile	ACT	T Thr	AAT	N Asn	AGT	S Ser
ATC	I Ile	ACC	T Thr	AAC	N Asn	AGC	S Ser
ATA	I Ile	ACA	T Thr	AAA	K Lys	AGA	R Arg
ATG	M Met i	ACG	T Thr	AAG	K Lys	AGG	R Arg
GTT	V Val	GCT	A Ala	GAT	D Asp	GGT	G Gly
GTC	V Val	GCC	A Ala	GAC	D Asp	GGC	G Gly
GTA	V Val	GCA	A Ala	GAA	E Glu	GGA	G Gly
GTG	V Val	GCG	A Ala	GAG	E Glu	GGG	G Gly

B.2 Bacterial

TTT	F Phe	TCT	S Ser	TAT	Y Tyr	TGT	C Cys
TTC	F Phe	TCC	S Ser	TAC	Y Tyr	TGC	C Cys

TTA	L Leu	TCA	S Ser	TAA	* Ter	TGA	* Ter
TTG	L Leu i	TCG	S Ser	TAG	* Ter	TGG	W Trp
CTT	L Leu	CCT	P Pro	CAT	H His	CGT	R Arg
CTC	L Leu	CCC	P Pro	CAC	H His	CGC	R Arg
CTA	L Leu	CCA	P Pro	CAA	Q Gln	CGA	R Arg
CTG	L Leu i	CCG	P Pro	CAG	Q Gln	CGG	R Arg
ATT	I Ile i	ACT	T Thr	AAT	N Asn	AGT	S Ser
ATC	I Ile i	ACC	T Thr	AAC	N Asn	AGC	S Ser
ATA	I Ile i	ACA	T Thr	AAA	K Lys	AGA	R Arg
ATG	M Met i	ACG	T Thr	AAG	K Lys	AGG	R Arg
GTT	V Val	GCT	A Ala	GAT	D Asp	GGT	G Gly
GTC	V Val	GCC	A Ala	GAC	D Asp	GGC	G Gly
GTA	V Val	GCA	A Ala	GAA	E Glu	GGA	G Gly
GTG	V Val i	GCG	A Ala	GAG	E Glu	GGG	G Gly

B.3 Yeast Mitochondrial

TTT	F Phe	TCT	S Ser	TAT	Y Tyr	TGT	C Cys
TTC	F Phe	TCC	S Ser	TAC	Y Tyr	TGC	C Cys
TTA	L Leu	TCA	S Ser	TAA	* Ter	TGA	W Trp
TTG	L Leu	TCG	S Ser	TAG	* Ter	TGG	W Trp
CTT	T Thr	CCT	P Pro	CAT	H His	CGT	R Arg
CTC	T Thr	CCC	P Pro	CAC	H His	CGC	R Arg
CTA	T Thr	CCA	P Pro	CAA	Q Gln	CGA	R Arg
CTG	T Thr	CCG	P Pro	CAG	Q Gln	CGG	R Arg
ATT	I Ile	ACT	T Thr	AAT	N Asn	AGT	S Ser
ATC	I Ile	ACC	T Thr	AAC	N Asn	AGC	S Ser
ATA	M Met i	ACA	T Thr	AAA	K Lys	AGA	R Arg
ATG	M Met i	ACG	T Thr	AAG	K Lys	AGG	R Arg
GTT	V Val	GCT	A Ala	GAT	D Asp	GGT	G Gly
GTC	V Val	GCC	A Ala	GAC	D Asp	GGC	G Gly
GTA	V Val	GCA	A Ala	GAA	E Glu	GGA	G Gly
GTG	V Val	GCG	A Ala	GAG	E Glu	GGG	G Gly

B.4 Vertebrate Mitochondrial

TTT	F Phe	TCT	S Ser	TAT	Y Tyr	TGT	C Cys
TTC	F Phe	TCC	S Ser	TAC	Y Tyr	TGC	C Cys
TTA	L Leu	TCA	S Ser	TAA	* Ter	TGA	W Trp
TTG	L Leu	TCG	S Ser	TAG	* Ter	TGG	W Trp
CTT	L Leu	CCT	P Pro	CAT	H His	CGT	R Arg
CTC	L Leu	CCC	P Pro	CAC	H His	CGC	R Arg

CTA	L	Leu		CCA	P	Pro	CAA	Q	Gln	CGA	R	Arg
CTG	L	Leu		CCG	P	Pro	CAG	Q	Gln	CGG	R	Arg
ATT	I	Ile	i	ACT	T	Thr	AAT	N	Asn	AGT	S	Ser
ATC	I	Ile	i	ACC	T	Thr	AAC	N	Asn	AGC	S	Ser
ATA	M	Met	i	ACA	T	Thr	AAA	K	Lys	AGA	*	Ter
ATG	M	Met	i	ACG	T	Thr	AAG	K	Lys	AGG	*	Ter
GTT	V	Val		GCT	A	Ala	GAT	D	Asp	GGT	G	Gly
GTC	V	Val		GCC	A	Ala	GAC	D	Asp	GGC	G	Gly
GTA	V	Val		GCA	A	Ala	GAA	E	Glu	GGA	G	Gly
GTG	V	Val	i	GCG	A	Ala	GAG	E	Glu	GGG	G	Gly

B.5 Mold Mitochondrial

TTT	F	Phe		TCT	S	Ser	TAT	Y	Tyr	TGT	C	Cys
TTC	F	Phe		TCC	S	Ser	TAC	Y	Tyr	TGC	C	Cys
TTA	L	Leu	i	TCA	S	Ser	TAA	*	Ter	TGA	W	Trp
TTG	L	Leu	i	TCG	S	Ser	TAG	*	Ter	TGG	W	Trp
CTT	L	Leu		CCT	P	Pro	CAT	H	His	CGT	R	Arg
CTC	L	Leu		CCC	P	Pro	CAC	H	His	CGC	R	Arg
CTA	L	Leu		CCA	P	Pro	CAA	Q	Gln	CGA	R	Arg
CTG	L	Leu	i	CCG	P	Pro	CAG	Q	Gln	CGG	R	Arg
ATT	I	Ile	i	ACT	T	Thr	AAT	N	Asn	AGT	S	Ser
ATC	I	Ile	i	ACC	T	Thr	AAC	N	Asn	AGC	S	Ser
ATA	I	Ile	i	ACA	T	Thr	AAA	K	Lys	AGA	R	Arg
ATG	M	Met	i	ACG	T	Thr	AAG	K	Lys	AGG	R	Arg
GTT	V	Val		GCT	A	Ala	GAT	D	Asp	GGT	G	Gly
GTC	V	Val		GCC	A	Ala	GAC	D	Asp	GGC	G	Gly
GTA	V	Val		GCA	A	Ala	GAA	E	Glu	GGA	G	Gly
GTG	V	Val	i	GCG	A	Ala	GAG	E	Glu	GGG	G	Gly

B.6 Invertebrate Mitochondrial

TTT	F	Phe		TCT	S	Ser	TAT	Y	Tyr	TGT	C	Cys
TTC	F	Phe		TCC	S	Ser	TAC	Y	Tyr	TGC	C	Cys
TTA	L	Leu		TCA	S	Ser	TAA	*	Ter	TGA	W	Trp
TTG	L	Leu	i	TCG	S	Ser	TAG	*	Ter	TGG	W	Trp
CTT	L	Leu		CCT	P	Pro	CAT	H	His	CGT	R	Arg
CTC	L	Leu		CCC	P	Pro	CAC	H	His	CGC	R	Arg
CTA	L	Leu		CCA	P	Pro	CAA	Q	Gln	CGA	R	Arg
CTG	L	Leu		CCG	P	Pro	CAG	Q	Gln	CGG	R	Arg
ATT	I	Ile	i	ACT	T	Thr	AAT	N	Asn	AGT	S	Ser
ATC	I	Ile	i	ACC	T	Thr	AAC	N	Asn	AGC	S	Ser

ATA	M Met i	ACA	T Thr	AAA	K Lys	AGA	S Ser
ATG	M Met i	ACG	T Thr	AAG	K Lys	AGG	S Ser
GTT	V Val	GCT	A Ala	GAT	D Asp	GGT	G Gly
GTC	V Val	GCC	A Ala	GAC	D Asp	GGC	G Gly
GTA	V Val	GCA	A Ala	GAA	E Glu	GGA	G Gly
GTG	V Val i	GCG	A Ala	GAG	E Glu	GGG	G Gly

B.7 Echinoderm Mitochondrial

TTT	F Phe	TCT	S Ser	TAT	Y Tyr	TGT	C Cys
TTC	F Phe	TCC	S Ser	TAC	Y Tyr	TGC	C Cys
TTA	L Leu	TCA	S Ser	TAA	* Ter	TGA	W Trp
TTG	L Leu	TCG	S Ser	TAG	* Ter	TGG	W Trp
CTT	L Leu	CCT	P Pro	CAT	H His	CGT	R Arg
CTC	L Leu	CCC	P Pro	CAC	H His	CGC	R Arg
CTA	L Leu	CCA	P Pro	CAA	Q Gln	CGA	R Arg
CTG	L Leu	CCG	P Pro	CAG	Q Gln	CGG	R Arg
ATT	I Ile	ACT	T Thr	AAT	N Asn	AGT	S Ser
ATC	I Ile	ACC	T Thr	AAC	N Asn	AGC	S Ser
ATA	I Ile	ACA	T Thr	AAA	N Asn	AGA	S Ser
ATG	M Met i	ACG	T Thr	AAG	K Lys	AGG	S Ser
GTT	V Val	GCT	A Ala	GAT	D Asp	GGT	G Gly
GTC	V Val	GCC	A Ala	GAC	D Asp	GGC	G Gly
GTA	V Val	GCA	A Ala	GAA	E Glu	GGA	G Gly
GTG	V Val	GCG	A Ala	GAG	E Glu	GGG	G Gly

B.8 Ascidian Mitochondrial

TTT	F Phe	TCT	S Ser	TAT	Y Tyr	TGT	C Cys
TTC	F Phe	TCC	S Ser	TAC	Y Tyr	TGC	C Cys
TTA	L Leu	TCA	S Ser	TAA	* Ter	TGA	W Trp
TTG	L Leu	TCG	S Ser	TAG	* Ter	TGG	W Trp
CTT	L Leu	CCT	P Pro	CAT	H His	CGT	R Arg
CTC	L Leu	CCC	P Pro	CAC	H His	CGC	R Arg
CTA	L Leu	CCA	P Pro	CAA	Q Gln	CGA	R Arg
CTG	L Leu	CCG	P Pro	CAG	Q Gln	CGG	R Arg
ATT	I Ile	ACT	T Thr	AAT	N Asn	AGT	S Ser
ATC	I Ile	ACC	T Thr	AAC	N Asn	AGC	S Ser
ATA	M Met	ACA	T Thr	AAA	K Lys	AGA	G Gly
ATG	M Met i	ACG	T Thr	AAG	K Lys	AGG	G Gly
GTT	V Val	GCT	A Ala	GAT	D Asp	GGT	G Gly
GTC	V Val	GCC	A Ala	GAC	D Asp	GGC	G Gly

GTA	V	Val	GCA	A	Ala	GAA	E	Glu	GGA	G	Gly
GTG	V	Val	GCG	A	Ala	GAG	E	Glu	GGG	G	Gly

B.9 *Flatworm Mitochondrial*

TTT	F	Phe	TCT	S	Ser	TAT	Y	Tyr	TGT	C	Cys
TTC	F	Phe	TCC	S	Ser	TAC	Y	Tyr	TGC	C	Cys
TTA	L	Leu	TCA	S	Ser	TAA	Y	Tyr	TGA	W	Trp
TTG	L	Leu	TCG	S	Ser	TAG	*	Ter	TGG	W	Trp
CTT	L	Leu	CCT	P	Pro	CAT	H	His	CGT	R	Arg
CTC	L	Leu	CCC	P	Pro	CAC	H	His	CGC	R	Arg
CTA	L	Leu	CCA	P	Pro	CAA	Q	Gln	CGA	R	Arg
CTG	L	Leu	CCG	P	Pro	CAG	Q	Gln	CGG	R	Arg
ATT	I	Ile	ACT	T	Thr	AAT	N	Asn	AGT	S	Ser
ATC	I	Ile	ACC	T	Thr	AAC	N	Asn	AGC	S	Ser
ATA	I	Ile	ACA	T	Thr	AAA	N	Asn	AGA	S	Ser
ATG	M	Met i	ACG	T	Thr	AAG	K	Lys	AGG	S	Ser
GTT	V	Val	GCT	A	Ala	GAT	D	Asp	GGT	G	Gly
GTC	V	Val	GCC	A	Ala	GAC	D	Asp	GGC	G	Gly
GTA	V	Val	GCA	A	Ala	GAA	E	Glu	GGA	G	Gly
GTG	V	Val	GCG	A	Ala	GAG	E	Glu	GGG	G	Gly

B.10 *Ciliate Nuclear*

TTT	F	Phe	TCT	S	Ser	TAT	Y	Tyr	TGT	C	Cys
TTC	F	Phe	TCC	S	Ser	TAC	Y	Tyr	TGC	C	Cys
TTA	L	Leu	TCA	S	Ser	TAA	Q	Gln	TGA	*	Ter
TTG	L	Leu	TCG	S	Ser	TAG	Q	Gln	TGG	W	Trp
CTT	L	Leu	CCT	P	Pro	CAT	H	His	CGT	R	Arg
CTC	L	Leu	CCC	P	Pro	CAC	H	His	CGC	R	Arg
CTA	L	Leu	CCA	P	Pro	CAA	Q	Gln	CGA	R	Arg
CTG	L	Leu	CCG	P	Pro	CAG	Q	Gln	CGG	R	Arg
ATT	I	Ile	ACT	T	Thr	AAT	N	Asn	AGT	S	Ser
ATC	I	Ile	ACC	T	Thr	AAC	N	Asn	AGC	S	Ser
ATA	I	Ile	ACA	T	Thr	AAA	K	Lys	AGA	R	Arg
ATG	M	Met i	ACG	T	Thr	AAG	K	Lys	AGG	R	Arg
GTT	V	Val	GCT	A	Ala	GAT	D	Asp	GGT	G	Gly
GTC	V	Val	GCC	A	Ala	GAC	D	Asp	GGC	G	Gly
GTA	V	Val	GCA	A	Ala	GAA	E	Glu	GGA	G	Gly
GTG	V	Val	GCG	A	Ala	GAG	E	Glu	GGG	G	Gly

B.11 Euplotid Nuclear

TTT	F	Phe	TCT	S	Ser	TAT	Y	Tyr	TGT	C	Cys
TTC	F	Phe	TCC	S	Ser	TAC	Y	Tyr	TGC	C	Cys
TTA	L	Leu	TCA	S	Ser	TAA	*	Ter	TGA	C	Cys
TTG	L	Leu	TCG	S	Ser	TAG	*	Ter	TGG	W	Trp
CTT	L	Leu	CCT	P	Pro	CAT	H	His	CGT	R	Arg
CTC	L	Leu	CCC	P	Pro	CAC	H	His	CGC	R	Arg
CTA	L	Leu	CCA	P	Pro	CAA	Q	Gln	CGA	R	Arg
CTG	L	Leu	CCG	P	Pro	CAG	Q	Gln	CGG	R	Arg
ATT	I	Ile	ACT	T	Thr	AAT	N	Asn	AGT	S	Ser
ATC	I	Ile	ACC	T	Thr	AAC	N	Asn	AGC	S	Ser
ATA	I	Ile	ACA	T	Thr	AAA	K	Lys	AGA	R	Arg
ATG	M	Met i	ACG	T	Thr	AAG	K	Lys	AGG	R	Arg
GTT	V	Val	GCT	A	Ala	GAT	D	Asp	GGT	G	Gly
GTC	V	Val	GCC	A	Ala	GAC	D	Asp	GGC	G	Gly
GTA	V	Val	GCA	A	Ala	GAA	E	Glu	GGA	G	Gly
GTG	V	Val	GCG	A	Ala	GAG	E	Glu	GGG	G	Gly

B.12 Alternative Yeast Nuclear

TTT	F	Phe	TCT	S	Ser	TAT	Y	Tyr	TGT	C	Cys
TTC	F	Phe	TCC	S	Ser	TAC	Y	Tyr	TGC	C	Cys
TTA	L	Leu	TCA	S	Ser	TAA	*	Ter	TGA	*	Ter
TTG	L	Leu	TCG	S	Ser	TAG	*	Ter	TGG	W	Trp
CTT	L	Leu	CCT	P	Pro	CAT	H	His	CGT	R	Arg
CTC	L	Leu	CCC	P	Pro	CAC	H	His	CGC	R	Arg
CTA	L	Leu	CCA	P	Pro	CAA	Q	Gln	CGA	R	Arg
CTG	S	Ser i	CCG	P	Pro	CAG	Q	Gln	CGG	R	Arg
ATT	I	Ile	ACT	T	Thr	AAT	N	Asn	AGT	S	Ser
ATC	I	Ile	ACC	T	Thr	AAC	N	Asn	AGC	S	Ser
ATA	I	Ile	ACA	T	Thr	AAA	K	Lys	AGA	R	Arg
ATG	M	Met i	ACG	T	Thr	AAG	K	Lys	AGG	R	Arg
GTT	V	Val	GCT	A	Ala	GAT	D	Asp	GGT	G	Gly
GTC	V	Val	GCC	A	Ala	GAC	D	Asp	GGC	G	Gly
GTA	V	Val	GCA	A	Ala	GAA	E	Glu	GGA	G	Gly
GTG	V	Val	GCG	A	Ala	GAG	E	Glu	GGG	G	Gly

B.13 Blepharisma Macronuclear

TTT	F	Phe	TCT	S	Ser	TAT	Y	Tyr	TGT	C	Cys
TTC	F	Phe	TCC	S	Ser	TAC	Y	Tyr	TGC	C	Cys

TTA	L	Leu	TCA	S	Ser	TAA	*	Ter	TGA	*	Ter
TTG	L	Leu	TCG	S	Ser	TAG	Q	Gln	TGG	W	Trp
CTT	L	Leu	CCT	P	Pro	CAT	H	His	CGT	R	Arg
CTC	L	Leu	CCC	P	Pro	CAC	H	His	CGC	R	Arg
CTA	L	Leu	CCA	P	Pro	CAA	Q	Gln	CGA	R	Arg
CTG	L	Leu	CCG	P	Pro	CAG	Q	Gln	CGG	R	Arg
ATT	I	Ile	ACT	T	Thr	AAT	N	Asn	AGT	S	Ser
ATC	I	Ile	ACC	T	Thr	AAC	N	Asn	AGC	S	Ser
ATA	I	Ile	ACA	T	Thr	AAA	K	Lys	AGA	R	Arg
ATG	M	Met i	ACG	T	Thr	AAG	K	Lys	AGG	R	Arg
GTT	V	Val	GCT	A	Ala	GAT	D	Asp	GGT	G	Gly
GTC	V	Val	GCC	A	Ala	GAC	D	Asp	GGC	G	Gly
GTA	V	Val	GCA	A	Ala	GAA	E	Glu	GGA	G	Gly
GTG	V	Val	GCG	A	Ala	GAG	E	Glu	GGG	G	Gly

Complete IDL C

C.1 File: DsLSRBioObjects.idl

```
//File: DsLSRBioObjects.idl
//  version: 29 October 2000.

#ifndef _DS_LSR_BIOOBJECTS_IDL_
#define _DS_LSR_BIOOBJECTS_IDL_

#pragma prefix "omg.org"

#include <CosLifeCycle.idl>
#include <CosPropertyService.idl>

module DsLSRBioObjects
{
    typedef sequence<string> StringList;
    typedef string Identifier;
    typedef sequence<Identifier> IdentifierList;

    enum StrandType {STRAND_NOT_KNOWN, STRAND_NOT_APPLICABLE,
        STRAND_PLUS, STRAND_MINUS, STRAND_BOTH};
    enum Basis {BASIS_NOT_KNOWN, BASIS_NOT_APPLICABLE,
        BASIS_EXPERIMENTAL, BASIS_COMPUTATIONAL, BASIS_BOTH};

#pragma version Basis 1.1

    valuetype Interval
    {
        public unsigned long  start;
        public unsigned long  length;
    };
```

```
valuetype SeqRegion : Interval
{
#pragma version SeqRegion 1.1

    public Identifier         id;
    public StrandType         strand_type;
    public boolean            start_relative_to_seq_end;
};

typedef sequence<SeqRegion> SeqRegionList;

valuetype CompositeSeqRegion : SeqRegion
{
    enum SeqRegionOperator
    {
        NONE,           // Region has no sub regions or the sub regions
                        // don't need special treatment.
        JOIN,           // Sub regions should be joined end-to-end to
                        // form a contiguous region.
        ORDER           // Sub region order is important.
    };

    public SeqRegionList      sub_regions;
    public SeqRegionOperator  region_operator;
};

interface Annotation : CosLifeCycle::LifeCycleObject
{
    readonly attribute string               name;  // type of annotation
    readonly attribute any                  value; // the annotation
    readonly attribute Basis                the_basis; // basis for annotation
    readonly attribute CosPropertyService::Properties  qualifiers;
};

typedef sequence<Annotation> AnnotationList;

exception IteratorInvalid
{
    string reason;
};

interface AnnotationIterator
{
    boolean      next(out Annotation the_annotation)
                        raises(IteratorInvalid);
    boolean      next_n(in unsigned long how_many,
                            out AnnotationList annotations)
                        raises(IteratorInvalid);
    void        reset();
    void        destroy();
};
```

```
interface SeqAnnotation : Annotation
{
    readonly attribute SeqRegion  seq_region;
};

typedef sequence<SeqAnnotation> SeqAnnotationList;

interface SeqAnnotationIterator
{
    boolean       next(out SeqAnnotation seq_annotation)
                          raises(IteratorInvalid);
    boolean       next_n(in unsigned long how_many,
                              out SeqAnnotationList seq_annotations)
                          raises(IteratorInvalid);
    void          reset();
    void          destroy();
};

exception IdentifierNotFound
{
    Identifier id;
};

exception IdentifierNotResolvable
{
    Identifier id;
    string reason;
};

exception IdentifierNotUnique
{
    Identifier id;
    IdentifierList ids;
};

exception IntervalOutOfBounds
{
    Interval invalid;
    Interval valid;
};

exception SeqRegionOutOfBounds
{
    SeqRegion invalid;
    Interval valid;
};

exception SeqRegionInvalid
{
    string reason;
    };

    exception NotUpdateable
    {
        string reason;
    };

    interface BioSequence : CosLifeCycle::LifeCycleObject
    {
#pragma version BioSequence 1.1

        readonly attribute string                   name;
        readonly attribute Identifier               id;
        readonly attribute string                   description;
        readonly attribute string                   seq;
        readonly attribute unsigned long            length;
        readonly attribute Basis                    the_basis;

        string          seq_interval(in Interval the_interval)
                            raises(IntervalOutOfBounds, SeqRegionInvalid);
        AnnotationList    get_annotations(
                                in unsigned long how_many,
                                in SeqRegion seq_region,
                                out AnnotationIterator the_rest)
                            raises(SeqRegionOutOfBounds, SeqRegionInvalid);
        unsigned long     num_annotations(in SeqRegion seq_region)
                            raises(SeqRegionOutOfBounds, SeqRegionInvalid);
        void            add_annotation(
                                in Annotation the_annotation)
                            raises(NotUpdateable, SeqRegionOutOfBounds,
                                SeqRegionInvalid);
    };

    typedef sequence<BioSequence> BioSequenceList;

    typedef sequence<unsigned long> UnsignedLongList;

    exception ReadingFrameInvalid
    {
        short invalid;
    };

    interface NucleotideSequence : BioSequence
    {
#pragma version NucleotideSequence 1.1

        readonly attribute boolean          circular;

        string        reverse_complement();
        string        reverse_complement_interval(in Interval the_interval)
                          raises(IntervalOutOfBounds, SeqRegionInvalid);
        string        translate_seq(
                              in short reading_frame,
                              out UnsignedLongList stop_locations)
                          raises(ReadingFrameInvalid);
        string        translate_seq_region(
                              in SeqRegion seq_region,
                              out UnsignedLongList stop_locations)
                          raises(SeqRegionOutOfBounds, SeqRegionInvalid);
    };

    typedef sequence<NucleotideSequence> NucleotideSequenceList;

    interface AminoAcidSequence : BioSequence
    {
#pragma version AminoAcidSequence 1.1
    };

    typedef sequence<AminoAcidSequence> AminoAcidSequenceList;

    typedef char    Residue;
    typedef char    Base;
    typedef Base    Codon[3];

    valuetype CodeRule
    {
        public Codon        the_codon;
        public Residue      the_residue;
    };

    typedef CodeRule        Coding[64];
    typedef string          GeneticCodeName;
    typedef sequence<GeneticCodeName> GeneticCodeNameList;

    exception InvalidResidue
    {
        Residue the_residue;
        unsigned long offset;
    };

    interface GeneticCode
    {
#pragma version GeneticCode 1.1

        readonly attribute Coding           the_coding;
        readonly attribute GeneticCodeName name;

        Residue translate_codon(in Codon the_codon)
            raises(InvalidResidue);
    };

    valuetype AlignmentElement
```

```
{
      public Object        element;
      public SeqRegion  seq_region;
      public string         key;
};

typedef sequence<AlignmentElement> AlignmentElementList;

interface AlignmentElementIterator
{
      boolean         next(out AlignmentElement element)
                         raises(IteratorInvalid);
      boolean         next_n(in unsigned long how_many,
                               out AlignmentElementList elements)
                         raises(IteratorInvalid);
      void           reset();
      void           destroy();
};

exception AlignmentObjectInvalid
{
      Object element;
      string reason;
};

exception ElementNotInAlignment
{
};

exception IndexOutOfBounds
{
      unsigned long invalid;
      Interval valid;
};

interface Alignment : CosLifeCycle::LifeCycleObject
{
#pragma version Alignment 1.1

      typedef string AlignType;
      typedef sequence<AlignType> AlignTypeList;

      const AlignType PROTEIN                  = "PROTEIN";
      const AlignType NON_PROTEIN          = "NON_PROTEIN";
      const AlignType SEQUENCE_ERROR = "SEQUENCE_ERROR";
      const AlignType UNKNOWN                 = "UNKNOWN";

      AlignmentElementList get_alignment_elements(
               in unsigned long start,
               in unsigned long how_many,
               out AlignmentElementIterator the_rest)
```

```
        raises(IndexOutOfBounds);

    unsigned long num_rows();
    unsigned long num_columns();

    SeqRegion get_seq_region(
            in string key,
            in Interval the_interval)
        raises(ElementNotInAlignment, IntervalOutOfBounds, SeqRegionInvalid);

    CompositeSeqRegion get_gaps(
            in string key,
            in Interval the_interval)
        raises(ElementNotInAlignment, IntervalOutOfBounds, SeqRegionInvalid);

    AlignType get_align_type_by_column(in unsigned long col)
        raises(IndexOutOfBounds);
};

typedef sequence<Alignment> AlignmentList;

interface Assembly : Alignment
{
};

valuetype SearchHit
{
    public Identifier                    id;
    public CosPropertyService::Properties  hit_info;
};

typedef sequence<SearchHit> SearchHitList;

interface SearchHitIterator
{
    boolean        next(out SearchHit hit)
                     raises(IteratorInvalid);
    boolean        next_n(in unsigned long how_many,
                          out SearchHitList hit_list)
                     raises(IteratorInvalid);
    void         reset();
    void         destroy();
};

valuetype SimilaritySearchHit : SearchHit
{
    public AlignmentList  alignment_list;
};

typedef sequence<SimilaritySearchHit> SimilaritySearchHitList;
```

```
interface BioSequenceIdentifierResolver
{
    BioSequence resolve(in Identifier id)
        raises (IdentifierNotFound, IdentifierNotResolvable,
            IdentifierNotUnique);
};

interface SearchResult :
    BioSequenceIdentifierResolver,
    CosLifeCycle::LifeCycleObject
{
    readonly attribute BioSequence  query_sequence;
    readonly attribute CosPropertyService::Properties  collection_info;
    StringList get_property_names();

    unsigned long  num_hits();

    SearchHitList  get_hits(
            in unsigned long start,
            in unsigned long how_many,
            out SearchHitIterator the_rest)
        raises (IndexOutOfBounds);
};

// optional interfaces

    interface AnnotationFactory
{
#pragma version AnnotationFactory 1.1

    Annotation create_annotation(
            in string name,
            in any value,
            in Basis the_basis,
            in CosPropertyService::Properties qualifiers);

    SeqAnnotation create_seq_annotation(
            in string name,
            in any value,
            in Basis the_basis,
            in CosPropertyService::Properties qualifiers,
            in SeqRegion seq_region)
        raises (SeqRegionInvalid);
};

exception SeqAnnotationOutOfBounds
{
    SeqAnnotation invalid;
    Interval valid;
};
```

```
    interface NucleotideSequenceFactory
    {
#pragma version NucleotideSequenceFactory 1.1

        NucleotideSequence create_sequence(
                in string name,
                in Identifier id,
                in string description,
                in string residues,
                in Basis the_basis,
                in boolean circular,
                in AnnotationList annotations)
            raises (InvalidResidue, SeqAnnotationOutOfBounds, SeqRegionInvalid);
    };

    interface AminoAcidSequenceFactory
    {
#pragma version AminoAcidSequenceFactory 1.1

        AminoAcidSequence create_sequence(
                in string name,
                in Identifier id,
                in string description,
                in string residues,
                in Basis the_basis,
                in AnnotationList annotations)
            raises (InvalidResidue, SeqAnnotationOutOfBounds, SeqRegionInvalid);
    };

    interface BioSequenceIterator
    {
        boolean       next(out BioSequence seq)
                        raises(IteratorInvalid);
        boolean       next_n(in unsigned long how_many,
                             out BioSequenceList seqs)
                        raises(IteratorInvalid);
        void          reset();
        void          destroy();
    };

    interface NucleotideSequenceIterator
    {
        boolean       next(out NucleotideSequence seq)
                        raises(IteratorInvalid);
        boolean       next_n(in unsigned long how_many,
                             out NucleotideSequenceList seqs)
                        raises(IteratorInvalid);
        void          reset();
        void          destroy();
    };
```

```
interface AminoAcidSequenceIterator
{
     boolean        next(out AminoAcidSequence seq)
                       raises(IteratorInvalid);
     boolean        next_n(in unsigned long how_many,
                             out AminoAcidSequenceList seqs)
                       raises(IteratorInvalid);
     void         reset();
     void         destroy();
};

exception InvalidGeneticCodeName
{
     string invalid_name;
};

interface GeneticCodeFactory
{
     const GeneticCodeName  STANDARD                        = "standard";
     const GeneticCodeName  BACTERIAL                       = "bacterial";
     const GeneticCodeName  YEAST_MITOCHONDRIAL             = "yeast
          mitochondrial";
     const GeneticCodeName  VERTEBRATE_MITOCHONDRIAL   = "vertebrate
          mitochondrial";
     const GeneticCodeName  MOLD_MITOCHONDRIAL              = "mold
          mitochondrial";
     const GeneticCodeName  INVERTEBRATE_MITOCHONDRIAL= "invertebrate
          mitochondrial";
     const GeneticCodeName  ECHINODERM_MITOCHONDRIAL= "echinoderm
          mitochondrial";
     const GeneticCodeName  ASCIDIAN_MITOCHONDRIAL   = "ascidian
          mitochondrial";
     const GeneticCodeName  FLATWORM_MITOCHONDRIAL= "flatworm
          mitochondrial";
     const GeneticCodeName  CILIATE_NUCLEAR              = "ciliate nuclear";
     const GeneticCodeName  EUPLOTID_NUCLEAR             = "euplotid nuclear";
     const GeneticCodeName  ALT_YEAST_NUCLEAR            = "alternative yeast
          nuclear";
     const GeneticCodeName  BLEPHARISMA_MACRONUCLEAR = "blepharisma
          macronuclear";

     readonly attribute GeneticCodeNameList genetic_code_names;
     GeneticCode create_genetic_code(in GeneticCodeName name)
          raises(InvalidGeneticCodeName);
};

interface CharacterAlignmentEncoder
{
     readonly attribute Alignment the_alignment;

     unsigned long  num_rows();                         // number of aligned
```

```
                                                        // objects. Delegate
        unsigned long  num_columns();                   // Delegate to Alignment

        string    get_name(in unsigned long row)        // first object is in row
                     raises(IndexOutOfBounds);          // one etc...
        StringList get_all_names();                     // all the Names

        string    get_cell_contents(in unsigned long row, in unsigned long col)
                     raises(IndexOutOfBounds);
        boolean    is_cell_a_gap(in unsigned long row, in unsigned long col)
                     raises(IndexOutOfBounds);
        unsigned long  get_cell_width(in unsigned long row, in unsigned long col)
                     raises(IndexOutOfBounds);
        unsigned long  max_column_width(in unsigned long col)
                     raises(IndexOutOfBounds);
        unsigned long  max_width();
    };

    interface SingleCharacterAlignmentEncoder : CharacterAlignmentEncoder
    {
#pragma version SingleCharacterAlignmentEncoder 1.1

        string       get_row(in unsigned long row)
                         raises(IndexOutOfBounds);
        string       get_row_interval(in unsigned long row, in Interval cols)
                         raises(IndexOutOfBounds, IntervalOutOfBounds,
                              SeqRegionInvalid);
        StringList   get_row_column_interval(in Interval rows, in Interval cols)
                         raises(IntervalOutOfBounds, SeqRegionInvalid);
        StringList   get_entire_alignment();        // probably the most common!
    };

    exception CannotEncodeAlignment
    {
        string reason;
    };

    interface CharacterAlignmentEncoderFactory
    {
        CharacterAlignmentEncoder create(in Alignment the_alignment)
            raises(CannotEncodeAlignment);
    };

    interface SingleCharacterAlignmentEncoderFactory
    {
        SingleCharacterAlignmentEncoder create(in Alignment the_alignment)
            raises(CannotEncodeAlignment);
    };
};

#endif // _DS_LSR_BIOOBJECTS_IDL_
```

C.2 File: DsLSRAnalysis.idl

```
//File: DsLSRAnalysis.idl
//  version: 29 October 2000.

#ifndef _DS_LSR_ANALYSIS_IDL_
#define _DS_LSR_ANALYSIS_IDL_

#pragma prefix "omg.org"

#include <orb.idl>
#include <CosPropertyService.idl>
#include <CosEventChannelAdmin.idl>
#include <CosLifeCycle.idl>
#include <TimeBase.idl>

module DsLSRAnalysis
{
    typedef sequence<string> StringList;

    valuetype AnalysisType
    {
        public string type;
        public string name;
        public string supplier;
        public string version;
        public string installation;
        public string description;
    };

    valuetype InputPropertySpec
    {
        public string           name;
        public CORBA::TypeCode  type;
        public boolean          mandatory;
        public any              default_value;
        public any              possible_values;
    };

    typedef sequence<InputPropertySpec> InputPropertySpecList;

    valuetype OutputPropertySpec
    {
        public string          name;
        public CORBA::TypeCode type;
    };

    typedef sequence<OutputPropertySpec> OutputPropertySpecList;

    enum AnalysisState
    {
        CREATED,    // Instance has been created but not yet executed.
        RUNNING,    // The analysis instance is running.
        COMPLETED,  // The instance has completed execution.
        TERMINATED_BY_REQUEST, // The instance was terminated by user request.
        TERMINATED_BY_ERROR    // The instance terminated due to an error.
    };

    valuetype AnalysisEvent
    {
        public string message;
    };

    valuetype StateChangedEvent : truncatable AnalysisEvent
    {
#pragma version StateChangedEvent 1.1

        public AnalysisState  previous_state;
        public AnalysisState  new_state;
    };

    valuetype HeartbeatProgressEvent : truncatable AnalysisEvent
    {
#pragma version HeartbeatProgressEvent 1.1
    };

    valuetype PercentProgressEvent : truncatable AnalysisEvent
    {
#pragma version PercentProgressEvent 1.1

        public float percentage;
    };

    valuetype TimeProgressEvent : truncatable AnalysisEvent
    {
#pragma version TimeProgressEvent 1.1

        public TimeBase::TimeT time_remaining;
    };

    valuetype StepProgressEvent : truncatable AnalysisEvent
    {
#pragma version StepProgressEvent 1.1

        public unsigned long total_steps;
        public unsigned long steps_completed;
    };

    interface AnalysisInstance;

    typedef string MetaData;
    exception DoesNotExistException { };

    interface AnalysisService
    {
        const string AnalysisTypeTag       = "TAG_ANALYSIS_TYPE";
        const string InputPropertiesTag    = "TAG_INPUT_PROPERTIES";
        const string OutputPropertiesTag = "TAG_OUTPUT_PROPERTIES";

        readonly attribute StringList metadata_tags;
        MetaData describe(in string tagname)
            raises (DoesNotExistException);

        readonly attribute AnalysisType type;
        readonly attribute InputPropertySpecList input_metadata;
        readonly attribute OutputPropertySpecList output_metadata;

        AnalysisInstance create_analysis (in CosPropertyService::Properties input)
            raises (CosPropertyService::MultipleExceptions);
    };

    exception NotRunnable { };
    exception NotRunning { };
    exception NotTerminated
    {
        string reason;
    };

    interface JobControl
    {
#pragma version JobControl 1.1

        readonly attribute TimeBase::UtcT created;
        readonly attribute TimeBase::TimeT elapsed;
        readonly attribute TimeBase::UtcT started;
        readonly attribute TimeBase::UtcT ended;

        void run()
            raises (NotRunnable, CosPropertyService::MultipleExceptions);
        void terminate()
            raises (NotRunning, NotRunnable, NotTerminated);
        void wait()
            raises (NotRunning, NotRunnable);
    };

    interface AnalysisInstance : CosLifeCycle::LifeCycleObject
    {
        readonly attribute AnalysisService      service;
        readonly attribute AnalysisState        status;
        readonly attribute CosEventChannelAdmin::EventChannel event_channel;
        readonly attribute AnalysisEvent        last_event;
        readonly attribute JobControl                    job_control;
        readonly attribute CosPropertyService::Properties     inputs;
        readonly attribute CosPropertyService::Properties     results;
        CosPropertyService::Properties get_result(in StringList name_list);
    };
};

#endif // _DS_LSR_ANALYSIS_IDL_
```

Domain Model DTD and XML D

D.1 File: DsLSRAnalysis.dtd

```
<!ELEMENT DsLSRAnalysis (analysis)+>

<!ELEMENT analysis (description?, input*, output*, analysis_extension?)>

<!ATTLIST analysis
      type             CDATA #REQUIRED
      name             CDATA #IMPLIED
      version          CDATA #IMPLIED
      supplier         CDATA #IMPLIED
      installation     CDATA #IMPLIED>

<!ELEMENT description ANY>

<!ELEMENT input (default?, allowed*, input_extension?)>

<!ATTLIST input
      type             CDATA #REQUIRED
      name             CDATA #REQUIRED
      mandatory        (true|false) "false">

<!ELEMENT default (#PCDATA)>
<!ELEMENT allowed (#PCDATA)>

<!ELEMENT output (output_extension?)>

<!ATTLIST output
      type             CDATA #REQUIRED
      name             CDATA #REQUIRED>

<!ENTITY % vendor_analysis_tags "<!ELEMENT analysis_extension EMPTY>">
<!ENTITY % vendor_input_tags    "<!ELEMENT input_extension EMPTY>">
<!ENTITY % vendor_output_tags   "<!ELEMENT output_extension EMPTY>">
```

```
%vendor_analysis_tags;
%vendor_input_tags;
%vendor_output_tags;
```

D.2 DsLSRBioAnalysis.xml

```
<?xml version = "1.0"?>
<!DOCTYPE DsLSRAnalysis SYSTEM "DsLSRAnalysis.dtd">

<DsLSRAnalysis>

    <analysis type = "search.list">
        <input
            name = "query_sequence"
            type = "IDL:omg.org/DsLSRBioObjects/BioSequence:1.0"
            mandatory = "true">
        </input>
        <input
            name = "sequence_list"
            type = "IDL:omg.org/DsLSRBioObjects/BioSequenceList:1.0"
            mandatory = "true">
        </input>
        <output
            name = "search_result"
            type = "IDL:omg.org/DsLSRBioObjects/SearchResult:1.0">
        </output>
    </analysis>

    <analysis type = "search.collection">
        <input
            name = "query_sequence"
            type = "IDL:omg.org/DsLSRBioObjects/BioSequence:1.0"
            mandatory = "true">
        </input>
        <input
            name = "sequence_iterator"
            type = "IDL:omg.org/DsLSRBioObjects/BioSequenceIterator:1.0"
            mandatory = "true">
        </input>
        <output
            name = "search_result"
            type = "IDL:omg.org/DsLSRBioObjects/SearchResult:1.0">
        </output>
    </analysis>

    <analysis type = "search.database">
        <input
            name = "query_sequence"
            type = "IDL:omg.org/DsLSRBioObjects/BioSequence:1.0"
            mandatory = "true">
        </input>
        <input
            name = "database_id"
            type = "IDL:omg.org/DsLSRBioObjects/Identifier:1.0"
```

```
                mandatory = "true">
            </input>
            <output
                name = "search_result"
type = "IDL:omg.org/DsLSRBioObjects/SearchResult:1.0">
            </output>
        </analysis>

        <analysis type = "search/annotation">
            <input
                name = "sequence"
                type = "IDL:omg.org/DsLSRBioObjects/BioSequence:1.0"
                mandatory = "true">
            </input>
            <output
                name = "sequence_annotation"
                type = "IDL:omg.org/DsLSRBioObjects/SeqAnnotationList:1.0">
            </output>
        </analysis>

        <analysis type = "search/region">
            <input
                name = "sequence"
                type = "IDL:omg.org/DsLSRBioObjects/BioSequence:1.0"
                mandatory = "true">
            </input>
            <output
                name = "sequence_region"
                type = "IDL:omg.org/DsLSRBioObjects/SeqRegionList:1.0">
            </output>
        </analysis>

        <analysis type = "search.list/similarity">
            <input
                name = "query_sequence"
                type = "IDL:omg.org/DsLSRBioObjects/BioSequence:1.0"
                mandatory = "true">
            </input>
            <input
                name = "sequence_list"
                type = "IDL:omg.org/DsLSRBioObjects/BioSequenceList:1.0"
                mandatory = "true">
            </input>
            <output
                name = "search_result"
                type = "IDL:omg.org/DsLSRBioObjects/SearchResult:1.0">
            </output>
        </analysis>

        <analysis type = "search.collection/similarity">
            <input
                name = "query_sequence"
                type = "IDL:omg.org/DsLSRBioObjects/BioSequence:1.0"
                mandatory = "true">
            </input>
        <input
            name = "sequence_iterator"
            type = "IDL:omg.org/DsLSRBioObjects/BioSequenceIterator:1.0"
            mandatory = "true">
        </input>
        <output
            name = "search_result"
            type = "IDL:omg.org/DsLSRBioObjects/SearchResult:1.0">
        </output>
    </analysis>

    <analysis type = "search.database/similarity">
        <input
            name = "query_sequence"
            type = "IDL:omg.org/DsLSRBioObjects/BioSequence:1.0"
            mandatory = "true">
        </input>
        <input
            name = "database_id"
            type = "IDL:omg.org/DsLSRBioObjects/Identifier:1.0"
            mandatory = "true">
        </input>
        <output
            name = "search_result"
            type = "IDL:omg.org/DsLSRBioObjects/SearchResult:1.0">
        </output>
    </analysis>

    <analysis type = "alignment.list">
        <input
            name = "sequence_list"
            type = "IDL:omg.org/DsLSRBioObjects/BioSequenceList:1.0"
            mandatory = "true">
        </input>
        <output
            name = "alignment"
            type = "IDL:omg.org/DsLSRBioObjects/Alignment:1.0">
        </output>
    </analysis>

    <analysis type = "alignment.collection">
        <input
            name = "sequence_iterator"
            type = "IDL:omg.org/DsLSRBioObjects/BioSequenceIterator:1.0"
            mandatory = "true">
        </input>
        <output
            name = "alignment"
            type = "IDL:omg.org/DsLSRBioObjects/Alignment:1.0">
        </output>
    </analysis>

    <analysis type = "alignment.list/assembly">
        <input
            name = "sequence_list"
        type = "IDL:omg.org/DsLSRBioObjects/BioSequenceList:1.0"
        mandatory = "true">
    </input>
    <output
        name = "assembly"
        type = "IDL:omg.org/DsLSRBioObjects/Assembly:1.0">
    </output>
</analysis>

<analysis type = "alignment.collection/assembly">
    <input
        name = "sequence_iterator"
        type = "IDL:omg.org/DsLSRBioObjects/BioSequenceIterator:1.0"
        mandatory = "true">
    </input>
    <output
        name = "assembly"
        type = "IDL:omg.org/DsLSRBioObjects/Assembly:1.0">
    </output>
</analysis>

<analysis type = "alignment/phylogeny">
    <input
        name = "alignment_list"
        type = "IDL:omg.org/DsLSRBioObjects/AlignmenList:1.0"
        mandatory = "true">
    </input>
    <output
        name = "alignment"
        type = "IDL:omg.org/DsLSRBioObjects/Alignment:1.0">
    </output>
</analysis>

<analysis type = "utility/molecular_weight">
    <input
        name = "sequence"
        type = "IDL:omg.org/DsLSRBioObjects/BioSequence:1.0"
        mandatory = "true">
    </input>
    <output
        name = "molecular_weight"
        type = "unsigned long">
    </output>
</analysis>

<analysis type = "utility/residue_composition">
    <input
        name = "sequence"
        type = "IDL:omg.org/DsLSRBioObjects/BioSequence:1.0"
        mandatory = "true">
    </input>
    <input
        name = "residue"
        type = "IDL:omg.org/DsLSRBioObjects/Residue:1.0"
        mandatory = "true">
    </input>
    <output
        name = "residue_composition"
        type = "double">
    </output>
</analysis>

<analysis type = "utility/ambiguous_residues">
    <input
        name = "sequence"
        type = "IDL:omg.org/DsLSRBioObjects/BioSequence:1.0"
        mandatory = "true">
    </input>
    <input
        name = "genetic_code"
        type = "IDL:omg.org/DsLSRBioObjects/GeneticCode:1.0"
        mandatory = "true">
    </input>
    <output
        name = "ambiguous_residues"
        type = "boolean">
    </output>
</analysis>

<analysis type = "utility/gc_content">
    <input
        name = "sequence"
        type = "IDL:omg.org/DsLSRBioObjects/NucleicAcidSequence:1.0"
        mandatory = "true">
    </input>
    <output
        name = "gc_content"
        type = "double">
    </output>
</analysis>

<analysis type = "utility/isoelectric_point">
    <input
        name = "sequence"
        type = "IDL:omg.org/DsLSRBioObjects/AminoAcidSequence:1.0"
        mandatory = "true">
    </input>
    <output
        name = "isoelectric_point"
        type = "double">
    </output>
</analysis>

<analysis type = "utility/translate_seq">
    <input
        name = "sequence"
        type = "IDL:omg.org/DsLSRBioObjects/NucleicAcidSequence:1.0"
        mandatory = "true">
    </input>
    <input
```

```
                name = "reading_frame"
                type = "short">
                    <default>-3</default>
                    <allowed>-2</allowed>
                    <allowed>-1</allowed>
                    <allowed>1</allowed>
                    <allowed>2</allowed>
                    <allowed>3</allowed>
            </input>
            <input
                name = "genetic_code"
                type = "IDL:omg.org/DsLSRBioObjects/GeneticCode:1.0"
                mandatory = "true">
            </input>
            <output
                name = "translated_seq"
                type = "string">
            </output>
        </analysis>

        <analysis type = "utility/translate_seq.seq_region">
            <input
                name = "sequence"
                type = "IDL:omg.org/DsLSRBioObjects/NucleicAcidSequence:1.0"
                mandatory = "true">
            </input>
            <input
                name = "sequence_region"
                type = "IDL:omg.org/DsLSRBioObjects/SeqRegion:1.0"
                mandatory = "true">
            </input>
            <input
                name = "genetic_code"
                type = "IDL:omg.org/DsLSRBioObjects/GeneticCode:1.0"
                mandatory = "true">
            </input>
            <output
                name = "translated_seq"
                type = "string">
            </output>
        </analysis>

    </DsLSRAnalysis>
```

Future Direction of Metamodel E

This specification uses metadata to describe analyses and inputs and outputs to analyses. Included in the specification is a DTD and example XML that shows the future direction of metadata within BSA. When more complex, more descriptive metadata is needed, the BSA metadata could be described using the mechanisms specified in the XMI. The sample better illustrates this idea.

E.1 File: DsLSRAnalysis - future.dtd

```
<!-- LSR BSA DTD -->

<!ENTITY % UmlMetaData SYSTEM "ad98-10-16.dtd">
%UmlMetaData;

<!ENTITY % DsLSRAnalysisXMI  SYSTEM "DsLSRAnalysisXMI.dtd">
%DsLSRAnalysisXMI;
```

E.2 File: DsLSRAnalysisXMI - future.dtd

```
<!-- LSR BSA Analysis Machinery DTD -->

<!ELEMENT DsLSRAnalysisXMI (analysis)+>
<!ATTLIST DsLSRAnalysisXMI
                              %XMI.element.att;
                              %XMI.link.att;
>


<!ELEMENT analysis (description?, input*, output*, XMI.extension*)>

<!ATTLIST analysis
                          analysisType  CDATA #REQUIRED
                          name        CDATA #IMPLIED
                          version             CDATA #IMPLIED
```

```
                    supplier        CDATA #IMPLIED
                    installation    CDATA #IMPLIED>

<!ELEMENT description (XMI.extension*)>

<!ELEMENT input (parameter*, XMI.extension*)>
<!ATTLIST input
                    name            CDATA #REQUIRED
                    mandatory       (true|false) "false">

<!ELEMENT output (parameter*, XMI.extension*)>
<!ATTLIST output
                    name            CDATA #REQUIRED>

<!ELEMENT parameter ((Foundation.Core.Parameter | logicalType), constraint*)>

<!ELEMENT logicalType (Foundation.Core.DataType | XMI.CorbaTypeCode | XMI.extension+)>

<!ELEMENT constraint (default?, allowed*, Foundation.Core.Constraint*, XMI.extension*)>

<!ELEMENT default (#PCDATA)>

<!ELEMENT allowed (#PCDATA)>
```

E.3 File: DsLSRBioAnalysis - future (sample).xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XMI SYSTEM 'DsLSRAnalysis-future.dtd'>

<XMI>

  <XMI.header>
<XMI.model xmi.name='sample' xmi.version='1.0'/>
<XMI.metamodel xmi.name='uml' xmi.version='1.1'/>
  </XMI.header>

  <XMI.content/>

  <XMI.extensions xmi.extender='omg.org/DsLSRAnalysis'>

<DsLSRAnalysisXMI>

      <analysis analysisType="similarity_analysis/database">

            <input name="query_sequence" mandatory="true">

              <parameter>

                  <Foundation.Core.Parameter>
                  <Foundation.Core.ModelElement.name>input</Foundation.Core.ModelElement.name>
                  <Foundation.Core.ModelElement.visibility xmi.value='public'/>
                  <Foundation.Core.Parameter.defaultValue>
                  <Foundation.Data_Types.Expression/>
      </Foundation.Core.Parameter.defaultValue>
      <Foundation.Core.Parameter.kind xmi.value='in'/>
      <Foundation.Core.Parameter.type>
      <Foundation.Core.Interface>
      <Foundation.Core.ModelElement.name>BioSequence</Foundation.Core.Model
          Element.name>
      <Foundation.Core.ModelElement.visibility xmi.value='public'/>
      <Foundation.Core.GeneralizableElement.isRoot xmi.value='false'/>
      <Foundation.Core.GeneralizableElement.isLeaf xmi.value='false'/>
      <Foundation.Core.GeneralizableElement.isAbstract xmi.value='false'/>
      </Foundation.Core.Interface>
      </Foundation.Core.Parameter.type>
      </Foundation.Core.Parameter>

  </parameter>

</input>



<input name="database_id" mandatory="true">

  <parameter>

      <Foundation.Core.Parameter>
      <Foundation.Core.ModelElement.name>input</Foundation.Core.ModelElement.name>
      <Foundation.Core.ModelElement.visibility xmi.value='public'/>
      <Foundation.Core.Parameter.defaultValue>
      <Foundation.Data_Types.Expression/>
      </Foundation.Core.Parameter.defaultValue>
      <Foundation.Core.Parameter.kind xmi.value='in'/>
      <Foundation.Core.Parameter.type>
      <Foundation.Core.Interface>
      <Foundation.Core.ModelElement.name>DbId</Foundation.Core.ModelElement.name>
      <Foundation.Core.ModelElement.visibility xmi.value='public'/>
      <Foundation.Core.GeneralizableElement.isRoot xmi.value='false'/>
      <Foundation.Core.GeneralizableElement.isLeaf xmi.value='false'/>
      <Foundation.Core.GeneralizableElement.isAbstract xmi.value='false'/></
          Foundation.Core.Interface>
      </Foundation.Core.Parameter.type>
      </Foundation.Core.Parameter>

      <constraint>
          <allowed> database1 </allowed>
          <allowed> database2 </allowed>
          <allowed> database3 </allowed>
          <allowed> database4 </allowed>
      </constraint>

  </parameter>


</input>
```

```
              <output name="hits">

                <parameter>

                    <Foundation.Core.Parameter>
                    <Foundation.Core.ModelElement.name>input</Foundation.Core.ModelElement.name>
                    <Foundation.Core.ModelElement.visibility xmi.value='public'/>
                    <Foundation.Core.Parameter.defaultValue>
                    <Foundation.Data_Types.Expression/>
                    </Foundation.Core.Parameter.defaultValue>
                    <Foundation.Core.Parameter.kind xmi.value='out'/>
                    <Foundation.Core.Parameter.type>
                    <Foundation.Core.Interface>
                    <Foundation.Core.ModelElement.name>Hits</Foundation.Core.ModelElement.name>
                    <Foundation.Core.ModelElement.visibility xmi.value='public'/>
                    <Foundation.Core.GeneralizableElement.isRoot xmi.value='false'/>
                    <Foundation.Core.GeneralizableElement.isLeaf xmi.value='false'/>
                    <Foundation.Core.GeneralizableElement.isAbstract xmi.value='false'/>
                    </Foundation.Core.Interface>
                    </Foundation.Core.Parameter.type>
                    </Foundation.Core.Parameter>

                </parameter>

              </output>


          </analysis>


      </DsLSRAnalysisXMI>

        </XMI.extensions>
      </XMI>
```

Glossary

Glossary entries are organized alphabetically.

Glossary Terms

Alignment	See *Sequence Alignment*
Ambiguity Code	Single character representation of an ambiguous nucleotide or residue.
Amino Acid	Any of a class of 20 small molecule building blocks that are combined to form proteins in living things (21 amino acids if selenocysteine is included). The sequence of amino acids in a protein and hence protein function are determined by the nucleotide sequence of its gene and the genetic code. The terms residue and amino acid are often used interchangeably.
Assembly	See *Sequence Assembly*
Base	See *Nucleotide*
Complementary Base	The nucleotide that chemically pairs up (hybridizes) with another nucleotide (called its complement) on the other strand, within a double-stranded sequence. G pairs with C in both DNA and RNA. A pairs with T in DNA. A pairs with U in RNA.
Complement	The sequence consisting of Complementary Bases.
Cladogram	See *Phylogenetic Tree*
Coding Sequence	A DNA sequence that contains appropriate start and stop codons, indicating the amino acid sequence translated from it could form a functional protein.
Codon	A set of three nucleotide bases in a DNA or RNA sequence, which together code for a unique amino acid. For example, the set AUG (adenine, uracil, guanine) codes for the amino acid methionine.

Contig or Contig Map	As used here, a graphical or data representation depicting the relative order of a linked library of small overlapping clones representing a complete chromosomal segment. See *Sequence Assembly.*
DNA (deoxyribonucleic acid)	The molecule that encodes genetic information. DNA is a double-stranded polymer of nucleotides. The two strands are held together by hydrogen bonds between base pairs of nucleotides. The four nucleotides in DNA contain the bases: adenine (A), guanine (G), cytosine (C), and thymine (T). In nature, base pairs form only between A and T and between G and C; thus the base sequence of each single strand can be deduced from that of its partner.
Expression	The conversion of the genetic instructions present in a DNA sequence into a unit of biological function in a living cell. Typically involves the process of transcription of a DNA sequence into an RNA sequence followed by translation of the RNA into protein. The RNA may be spliced before translation to remove introns.
Exon	Segment of a (genomic) sequence that is translated into a segment of a protein. See also *Intron*.
Gap	The opening and addition of one or more spaces to individual sequences in an alignment, in order to increase the consensus of the overall mapping. A gap represents a failure to establish equivalence between nucleotides in a particular region of a sequence when aligning it with one or more other sequences.
Gene	A length of DNA which codes for a particular protein, or in certain cases a functional or structural RNA molecule. Genes may be inferred from the DNA sequence by way of a coding sequence.
Genetic Code	The full set of codons in DNA or mRNA. Each codon is made up of three nucleotides which call for a unique amino acid. For example, the set AUG (adenine, uracil, guanine) calls for the amino acid methionine in the standard genetic code. The sequence of codons along an mRNA molecule specifies the sequence of amino acids in a particular protein.
Genome	The complete set of genetic information for a particular organism.
Genomic	Pertaining to or contained within a genome; also: chromosomal.
Hidden Markov Model (HMM)	A stochastic generative model for a series defined by a finite set of states, a discrete alphabet of symbols, a probability transition matrix, and a probability emission matrix.
Intron	Segment of the (genomic) sequence that is removed (spliced) from the RNA molecule prior to translation. Introns are therefore not translated to protein in a living cell.
Non-Coding	A class of genomic sequence that is not translated into a protein sequence. Non-coding sequence consists of introns and intergenic regions that may contain "junk" DNA such as repeat sequences.

Nucleic Acid	A polymer of nucleotides. DNA and RNA are different classes of nucleic acids. May be double- or single-stranded.
Nucleotide	A subunit of DNA or RNA consisting of a nitrogenous base (adenine, guanine, thymine, or cytosine in DNA; adenine, guanine, uracil, or cytosine in RNA), a phosphate molecule, and a sugar molecule (deoxyribose in DNA and ribose in RNA).
Phylogenetic Tree	A map, dendrogram, cladogram, or other data or graphical representation of a Phylogeny.
Phylogeny (phylogenesis, phylogenetic, phylogenic)	The evolutionary history of a particular taxonomic group, usually a species.
Profile	A table that lists the frequencies of finding each of the 20 amino acids at each position in conserved sequence pattern; used in sensitive sequence searches.
Protein	A biological molecule that consists of many amino acids chained together by peptide bonds. The sequence of amino acids in a protein is determined by the sequence of nucleotides in a DNA molecule. Proteins perform most of the enzymatic and structural roles within living cells.
RNA (ribonucleic acid)	A class of nucleic acids that consist of nucleotides containing the bases: adenine (A), guanine (G), cytosine (C), and uracil (U). An RNA molecule is typically single-stranded and can pair with DNA (where U pairs with A) or with another RNA molecule. RNA nucleotides are chemically distinct from DNA nucleotides and enable RNA molecules to have more complex structural and functional roles within a living cell.
Reading Frame	The ‘phase’ of the starting point of a translation. As each codon consists of three bases, a translation of a nucleotide sequence will yield entirely different protein sequences depending on this. Negative values are often used to denote translation of the reverse strand.
Residue	Amino acid; sometimes: nucleotide.
Reverse Complement	The sequence obtained by reading the opposite (complementary) strand of a nucleic acid sequence in the reverse direction.
Sequence	The order of nucleotides in a DNA or RNA molecule, or the order of amino acids in a protein.
Sequence Alignment	The explicit mapping between the residues of two or more sequences. A sequence alignment may have gaps. Alignments are used to establish similarities between sequences and/or sequence families.

Sequence Assembly	A series of linked sequence alignment analysis steps that is used for constructing a contig.
Splicing	The removal of introns from an RNA sequence leaving only the exons which are then translated into a protein.
Translation	The conversion of a nucleic acid sequence into an amino acid sequence according to the rules of a genetic code.