BIND SOAP Manual

Software Version: 1.0
Specification Version: 3.2

Written By:
Marc R. Dumontier
Shawn Konopinsky
Charles Roy
John J. Salama

Download the most recent copy of this document at
http://soap.bind.ca/BINDSOAPManual.pdf

March 2005

Table of Contents

1. Introduction

The **B**iomolecular **I**nteraction **N**etwork **D**atabase (BIND) is a publicly funded, publicly available database (http://www.bind.ca/). The database contains a collection of records documenting molecular interactions. The contents of BIND include high-throughput data submissions and hand-curated information gathered from the scientific literature.

This manual describes a language independent, programmable interface to the contents of the BIND database through the use of web services technology.

1.1 More About BIND

BIND is an interaction database with three classifications for molecular associations: molecules that associate with each other to form interactions, molecular complexes that are formed from one or more interaction(s) and pathways that are defined by a specific sequence of two or more interactions.

A BIND record represents an interaction between two or more objects that is believed to occur in a living organism. A biological object can be a protein, DNA, RNA, ligand, molecular complex, gene, photon or an unclassified biological entity. BIND records are created for interactions which have been shown experimentally and published in at least one peer-reviewed journal. A record also references any papers with experimental evidence that support or dispute the associated interaction.

Interactions are the basic units of BIND and can be linked together to form molecular complexes or pathways.

A molecular complex is a collection of two or more molecules that associate to form a functional unit in a living organism. In BIND, complexes are represented as molecular complex objects, formed by linking two or more interaction records. Molecular complex records are supplemented with additional information such as complex topology and the number of subunits (BIND objects) involved in the interaction.

A pathway is a collection of two or more interactions that occur in a defined sequence within a living organism. In BIND, pathways are represented by Pathway records that are formed by linking two or more interaction records. Pathway records are supplemented with additional information such as which stage of the cell cycle the pathway exists and whether the pathway is associated with a particular disease.

The object-oriented design of BIND allows the database to represent a diversity of interactions in a format that is efficiently interpreted by computers and used by software engineers developing novel Bioinformatics tools. Thanks to submissions from Blueprint curators and external researchers, BIND has been growing exponentially since it inception in 1998. As BIND grows and receives submissions from researchers in different countries, researchers may use BIND to understand publicly available experimental data in a global context.

1.2 What is BIND SOAP?

BIND SOAP is a web service providing end users with the ability to access functionality offered by the BIND Search Service through a remote **A**pplication **P**rogramming **I**nterface (API). This means that users can access and compute on BIND interaction, molecular complex, and pathway data directly within their application. Since the web service is based on SOAP, users are not limited in choice of platform or object oriented language used to develop their application.

Furthermore, depending on the object oriented language chosen, users can find freely available applications that can auto-generate client stub code given a WSDL file. The BIND SOAP WSDL file is provided at http://soap.bind.ca/wsdl/bind.wsdl.

For more information on methods provided in the BIND SOAP API, please refer to Section 4 of this document, or visit the online version at http://soap.bind.ca/help_soap_api.jsp.

For more information on BIND, please refer to the About BIND page <http://www.blueprint.org/bind/bind_about.html>.

1.3 A Brief Overview of SOAP

Simple **O**bject **A**ccess **P**rotocol is a W3C recommendation, based on the XML protocol, defining the exchange of information between applications over the Internet. The protocol provides an extensible way for applications to communicate with one another regardless of operating system and development language. Server side applications employing SOAP are referred to as web services, and use the **W**eb **S**ervices **D**escription **L**anguage (WSDL) standard to define the service to client applications.

For more information on SOAP, please refer to the W3C SOAP v1.2 Document <http://www.w3.org/TR/soap>.

For more information on WSDL, please refer to the W3C WSDL v1.1 Document <http://www.w3.org/TR/wsdl>.

1.4 WSDL Explained

WSDL stands for **W**eb **S**ervices **D**escription **L**anguage. WSDL is an XML based language used to describe web services. It specifies the location of the service and the operations, or methods, the service exposes. In addition, many toolkits can use a WSDL file to aid in writing SOAP clients and SOAP servers with little additional code needed by the end developer. More information on auto generation of client stub code can be found in Section 3 of this manual.

The major elements in a WSDL document are as follows:

Element	**Defines**
<portType>	The operations performed by the web service
<message>	The messages used by the web service
<types>	The data types used by the web service
<binding>	The communication protocols used by the web service

These elements are structured in a specific manner (see below) to describe the location of a web service and the methods offered by the service.

The main structure of a WSDL document looks like this:

```
<definitions>
<types>
  definition of types.....
</types>

<message>
  definition of a message....
</message>

<portType>
  definition of a port.....
</portType>

<binding>
  definition of a binding....
</binding>
</definitions>
```

1.5 Online Resources

1.5.1 SOAP

http://www.w3.org/TR/soap - Latest version of the SOAP 1.2 specification as submitted to the World Wide Web Consortium.

http://www.w3schools.com/soap/default.asp - A good first read for those just getting acquainted with SOAP. Includes all the information needed to understand the basics of a SOAP service.

http://www.onjava.com/pub/a/onjava/excerpt/java_xml_2_ch2/ - The first place to look if you already understand how SOAP works, and you will be working in Java. This tutorial includes some nice examples to explain how to easily write client-side code in Java to communicate with a SOAP service.

http://www.topxml.com/soap/ - Has lots of good information for .NET, Perl, and Java client implementations. Good reference, and not so helpful to read from start to finish.

1.5.2 WSDL

http//www.w3.org/TR/wsdl – Latest version of the WSDL 1.1 specification as submitted to the World Wide Web Consortium.

http://developers.sun.com/sw/building/tech_articles/overview_wsdl.html - Short and sweet description of the structure of a WSL document and how it fits into the SOAP service architecture.

http://www.w3schools.com/wsdl/wsdl_intro.asp - A good first read to grasp the scope of what WSDL is and how it can minimize the amount of client code needed and maximum program correctness when properly used.

http://www-106.ibm.com/developerworks/library/ws-intwsdl - A more in-depth read on WSDL. Contains many good examples and further links for WSDL resources.

2. SOAP in More Detail: Attachments

BIND SOAP returns data, which at times can very large in size (in excess of one gigabyte). Generally, SOAP toolkits do not allow for streaming the results. The result of this means that when the client receives a response message that it must build up the entire message in memory before any further actions can be taken on the response (i.e. writing the resulting BIND records to a file, or accessing the XML information within using DOM or SAX). The obvious downfall of this comes when receiving a message with a total size exceeding the memory available to the client program – an out of memory error will surely result. Utilizing the fact that binary and textual files can be effectively 'attached' outside the body of the response SOAP message package can avert this problem. (See 1.4 SOAP Resources for more info).

A SOAP message might need to be transmitted together with attachments of various sorts, in addition any information contained explicitly within the SOAP message package. For example, when sending an email, it is possible to 'attach' a GIF or JPEG image to send in addition to the message body. These attachments can be in either text or binary format. SOAP messages can be associated in the same way with one or more attachments in their native format using either a MIME or DIME structure for transport. BIND SOAP supports sending data as either MIME or DIME attachments.

SOAP messages with attachments are constructed using the "Multipart/Related" media type. The MIME "Multipart/Related" encapsulation of a SOAP message means that there are more than one 'part' to the entire response message. The first part is the actual SOAP message package (in XML), and any subsequent parts contain the attachments. DIME is Microsoft's answer to MIME, and works in very much the same way. The message is wrapped in a binary header describing the lengths of the attachments. We have not seen any significant speed differences between these two attachment types. Please see section 3.x to determine which implementation languages/clients support attachments.

2.1 Attachment Resources

MIME - http://www.w3.org/TR/SOAP-attachments
DIME - http://msdn.microsoft.com/msdnmag/issues/02/12/DIME/

3. Getting Started with BIND SOAP

3.1 Choosing a Client Side Language

One of the major benefits of using a SOAP service is the user is not limited in the choice of platform or language used to program the client. But with this choice comes confusion on the best language to use.

Obviously, choosing the most appropriate programming language will depend on restrictions that may be in place in your organization with respect to programming languages or the platforms being used by your organization. For example if your organization does not contain any Windows boxes, then you could not realistically build a BIND SOAP client in VB.NET.

However, not so obvious are advantages offered by each language as implemented in their respective SOAP package/module. Since SOAP, SOAP with Attachments and WSDL are relatively new technologies, not all major languages have implemented full support for all aspects of the aforementioned specifications. The three technical areas to consider in deciding which language to use are:

1. Auto generation of client side stub code.
2. Support for attachments
3. Support for streaming attachments

The languages considered for the purpose of this manual and client code examples are as follows:

Language	SOAP Package/Module	Availability
Java	Axis	http://ws.apache.org/axis
C/C++	gSoap	http://www.cs.fsu.edu/~engelen/soap.html
Perl	SOAP::Lite	http://www.soaplite.com
Python	SOAPpy	http://pywebsvcs.sourceforge.net/
VisualBasic	.NET	http://www.microsoft.com/downloads/details.aspx?FamilyID=262d25e3-f589-4842-8157-034d1e7cf3a3&displaylang=en

3.2 Using JAVA

The Java example client uses Apache AXIS 1.2 to communicate with BIND SOAP. AXIS is an open source web services toolkit freely available from Apache (http://ws.apache.org/axis). AXIS is included in this distribution under the "lib" folder. You may replace the jars in the lib folder with a different version of AXIS if you wish.

Included in this distribution is a build file that uses ANT, also available from Apache (http://ant.apache.org/).

To run the demo:

1. Unpack the archive
2. If you do not have ANT and want to use the build file, install ANT.
3. Build the source (and generate stub from the WSDL file) with ANT:
   'ant build'
4. Run the demo with ANT:
   'ant run'

AXIS provides a WSDL2Java tool that uses BIND's published WSDL file in order to generate 'stub' code for use in the Java client. The end result is that you can call the methods on the stub code without worrying about SOAP specifics. The ANT build file uses this tool during compilation to ensure the 'stub' files are generated before attempting to run the client.

When writing clients, the generated stub code abstracts most of the details of the web services. A "BINDSOAPBindingStub" object is created (See BINDSOAPDemo below). Using this object, you may call BINDSOAP services. See the supporting in Section 4.x of this documentation for the particulars of BINDSOAP's service methods available from the BINDSOAPBindingStub.

The BINDSOAPDemo class is the client code that creates and calls on functions of the 'BINDSOAPBindingStub'. The BINDSOAPDemo source is located in the 'java\src\org\blueprint\webservices\bind\soap\client' directory of the provided zip-file. By inspecting the BINDSOAPDemo code, it is clear how simple queries to BIND SOAP can be.

AXIS supports all methods (including attachments) provided by the BIND SOAP service. If you anticipate that the data being returned is too large to hold in memory, we suggest that you use attachments. As stated above, when not using attachments, the entire data result is held in memory to be verified before it is possible to dump it out to file or otherwise manipulate it.

3.3 Using C/C++

This example client demonstrates using the BIND SOAP service from the C language. The toolkit used is gSOAP and is available from http://www.cs.fsu.edu/~engelen/soap.html.

To run the demo:

1. Unpack the archive
2. Install gSOAP
3. Edit the Makefile variable GSOAP_HOME to indicate the location where gSoap is installed
4. Run the Makefile:
   'make'
5. Run the client program:
   './bindsoap'

NOTE: this code was written using gSOAP version 2.7-linux

gSoap provides a wsdl2h tool that is used with the bind.wsdl file for BINDSOAP in order to generate 'stub' code for use in the C/C++ client. The Makefile uses this tool during compilation to ensure the 'stub' files are generated before attempting to run the client.

The gSoap toolkit supports all methods (including attachments) provided by the BINDSOAP service. It is important to note however that gSoap only "streams" DIME to files, MIME attachments are kept in memory while being received, thus negating the purpose of using attachments. Therefore when using large datasets with this toolkit, we recommend using DIME attachments.

3.4 Using Perl

This example was written using the SOAP::Lite modules available for free from http://www.soaplite.com.

The Perl example demonstrates how to setup a Client-side service proxy object that uses the BINDSOAP WSDL file. It then uses the service proxy to call BINDSOAP web services.

To run the demo:

1. Unpack the archive
2. Ensure you have Perl installed
3. If you don't have it already, install SOAP::Lite
4. Execute the Perl client by running:
   'perl bindsoap.pl' from the directory containing the file

At the time of this release the SOAP::Lite stable release (v0.60) did not fully support MIME or DIME attachments, hence we excluded those calls from our Perl test client (bindsoap1.pl). However, the current development release of SOAP::Lite (v0.65) does support MIME attachments. It is important to note however that SOAP::Lite v0.65 does not stream MIME attachments. The attachments are kept in memory while being received, thus negating the purpose of using attachments.

We've included bindsoap2.pl which shows how to use SOAP::Lite with MIME attachments.

3.5 Using Python

The Python example uses SOAPpy, an open source Python implementation. It is freely available from http://pywebsvcs.sourceforge.net/. The client included in this package uses SOAPpy to create a client SOAPProxy object pointing to BINDSOAP. It then uses this object to call BINDSOAP methods. In contrast to the Java and Perl clients, the Python demo client we have provided does NOT use the WSDL file. Rather, the bindsoap.py client uses the URL of the SOAP service and namespace to create an object on which to call the service methods.

To run the demo:

1. Ensure you have the latest stable version of Python installed (http://www.python.org/)
2. If you don't have it already, install the SOAPpy module (http://pywebsvcs.sourceforge.net/)
3. Execute the Python client by running 'python bindsoap.py' from the directory containing the file.

See the supporting in Section 4.x of this documentation for the particulars of BINDSOAP's service available methods. At the time of this release, SOAPpy does not fully support MIME or DIME attachments; hence we excluded those calls from our Python test client.

3.6 Using VB.NET

This example was written using the Microsoft .NET Framework v1.1 available for free at:
http://www.microsoft.com/downloads/details.aspx?FamilyID=262d25e3-f589-4842-8157-034d1e7cf3a3&displaylang=en.

Included in this distribution is a solution file to build the project using Microsoft Visual Studios .NET. For more information on Visual Studios .NET, please visit http://msdn.microsoft.com/vstudio/.

Note that if you already have Microsoft Visual Studios .NET installed, you will already have the Microsoft .NET Framework installed. However, it's important to check which version of the .NET Framework is installed on your system.

The Visual Basic .NET example demonstrates how to setup a client side service proxy object that uses the BINDSOAP WSDL file. It then uses the service proxy to call BINDSOAP web services and display the output in a rich text box object contained in a window form object.

To run the demo:

1. Unpack the archive
2. Ensure you have Microsoft .NET Framework v1.1 installed
3. Double click on BINDSOAP_VB.exe located in the bin directory of the resulting unpacked archive
4. Press the 'Run Examples' button located at the top right hand corner of the BINDSOAP Example form.

The .NET Framework Software Development Kit provides a tool called wsdl.exe to auto generate VB, C# or J# 'stub' code given the BIND WSDL file. Typically this tool will be located at %SYSTEMDRIVE%\Program Files\Microsoft.NET\SDK\v1.1\Bin, but if you're unable to locate it using the above path, use the Windows Search interface to find the application.

For this code example, wsdl.exe was used with the following arguments to auto generate the stub VB code used to access the BINDSOAP service:

> wsdl /language:vb http://soap.bind.ca/wsdl/bind.wsdl

At the time of this release, we were not able to determine proper support for MIME or DIME attachments using v1.1 of the Microsoft .NET Framework. That said, Microsoft states that there is full support for SOAP with Attachments using their Web Services Enhancements v2.0 (WSE 2.0) < http://msdn.microsoft.com/webservices/building/wse/>.

Included in the BINDSOAP_VB client example code is a class called BINDService_WSE.vb that was auto generated using a tool called WseWsdl2.exe available with the WSE 2.0 SP3 release. Typically this tool will be located at %SYSTEMDRIVE\Program Files\Microsoft WSE\v2.0\Wsdl, but if you're unable to locate it using the above path, use the Windows Search interface to find the application.

For this code example WseWsdl2.exe was used with the following arguments to auto generate the stub VB code use to access the BINDSOAP service:

> WseWsdl2 http://soap.bind.ca/wsdl/bind.wsdl vb

Once again, although this auto generated class is provided in the client code example, there is no examples of how to use or implement the methods generated for the purposes of MIME or DIME attachments.

4. BIND SOAP Methods

A live API of your service methods are available at: http://soap.bind.ca/help_soap_api.jsp

The following methods return a string as a result of calling the method: isServiceAlive, getSoftwareVersion, getSpecVersion. All other methods return SearchResultBeans objects (or structs, dependant on the language of implementation) as a result of a successful call. The structure of a SearchResultBean is as follows:

Field	Description
TotalRecordsFound	Number of records returned
Records	The actual export is saved here
Query	The query used to produce this bean
ReturnType	The return type

4.1 isServiceAlive

4.1.1 Overview

The 'isServiceAlive' function returns a boolean value indicating if the BIND SOAP web service is up and running.

4.1.2 Parameters

None

4.1.3 Returns

Boolean - Returns true if the web service is running, false if it is not running.

4.1.4 Possible Exceptions

None.

4.2 getSoftwareVersion

4.2.1 Overview

Returns the current version number of the BIND SOAP web service being queried.

4.2.2 Parameters

None

4.2.3 Returns

String - The version number of the web service.

4.2.4 Possible Exceptions

A BINDSOAPException is thrown if there was a problem determining the software version.

4.3 getSpecVersion

4.3.1 Overview

Returns the BIND specification version implemented by the web service.

4.3.2 Parameters

None

4.3.3 Returns

String - The BIND specification version implemented.

4.3.4 Possible Exceptions

A BINDSOAPException is thrown if there was a problem determining the spec version.

4.4 idSearch

4.4.1 Overview

Given an identifier and the type of identifier returns all BIND records found, in the format specified.

4.4.2 Parameters

Name	Type	Description
Id	String	Value of query identifier
IdType	String	One of the following strings: 1. ‘bindid’ 2. 'gi' 3. 'entrezgene' 4. 'pmid' 5. 'taxid' 6. 'smallmoleculeid' 7. 'afcs' 8. 'beilstein' 9. 'cas' 10. 'cdd' 11. 'cog' 12. 'colibase' 13. 'ddbj'

		14. 'dictybase' 15. 'einecs' 16. 'embl' 17. 'emd' 18. 'ensembl' 19. 'flybase' 20. 'genbank' 21. 'GO' 22. 'interpro' 23. 'ipi' 24. 'klotho' 25. 'locuslink' 26. 'mdl' 27. 'merck' 28. 'MGI' 29. 'mmcif' 30. 'mmdb' 31. 'omim' 32. 'pdb' 33. 'pfam' 34. 'pir' 35. 'refseq' 36. 'rgd' 37. 'sgd' 38. 'smart' 39. 'swissprot' 40. 'tair' 41. 'atd' 42. 'cmr' 43. 'trembl' 44. 'unigene' 45. 'uniprot' 46. 'wormbase' 47. 'zfin'
returnType	String	One of the following strings: 1. 'idlist' 2. 'gipair' 3. 'go' 4. 'domain' 5. 'fasta' 6. 'cytoscape' 7. 'xml' 8. 'submitxml'

		9. 'submitasn' 10. 'flat' 11. 'psi'

4.4.3 Returns

SearchResultBean - A SearchResultBean containing the all BIND records found, in the format specified.

4.4.4 Possible Exceptions

A BINDSOAPException is thrown if there is unexpected input, such as an unknown IdType or unknown returnType. This exception will also be thrown in the instance that there was an excepted error attempting to send the results.

4.5 idSearchAttachment

4.5.1 Overview

Given an identifier and the type of identifier returns all BIND records found, in the format specified. The developer also has the option of restricting the search by BIND record type.

4.5.2 Parameters

Name	Type	Description
Id	String	Value of query identifier
IdType	String	One of the following strings: 1. 'bindid' 2. 'gi' 3. 'entrezgene' 4. 'pmid' 5. 'taxid' 6. 'smallmoleculeid' 7. 'afcs' 8. 'beilstein' 9. 'cas' 10. 'cdd' 11. 'cog' 12. 'colibase' 13. 'ddbj' 14. 'dictybase' 15. 'einecs' 16. 'embl' 17. 'emd'

		18. 'ensembl' 19. 'flybase' 20. 'genbank' 21. 'GO' 22. 'interpro' 23. 'ipi' 24. 'klotho' 25. 'locuslink' 26. 'mdl' 27. 'merck' 28. 'MGI' 29. 'mmcif' 30. 'mmdb' 31. 'omim' 32. 'pdb' 33. 'pfam' 34. 'pir' 35. 'refseq' 36. 'rgd' 37. 'sgd' 38. 'smart' 39. 'swissprot' 40. 'tair' 41. 'atd' 42. 'cmr' 43. 'trembl' 44. 'unigene' 45. 'uniprot' 46. 'wormbase' 47. 'zfin'
ReturnType	String	One of the following strings: 1. 'idlist' 2. 'gipair' 3. 'go' 4. 'domain' 5. 'fasta' 6. 'cytoscape' 7. 'xml' 8. 'submitxml' 9. 'submitasn' 10. 'flat' 11. 'psi'

AttachmentType	String	One of the following strings: 1. 'mime' 2. 'dime'

4.5.3 Returns

SearchResultBean - A SearchResultBean containing the all BIND records found, in the format specified.

4.5.4 Possible Exceptions

A BINDSOAPException is thrown if there is unexpected input, such as an unknown IdType or unknown returnType. This exception will also be thrown in the instance that there was an excepted error attempting to send the results.

4.6 textSearch

4.6.1 Overview

Given a Lucene query string, returns all BIND records found, in the format specified.

4.6.2 Parameters

Name	Type	Description
Query	String	The Lucene query string (see Section 5 of this manual).
ReturnType	String	One of the following strings: 1. 'idlist' 2. 'gipair' 3. 'go' 4. 'domain' 5. 'fasta' 6. 'cytoscape' 7. 'xml' 8. 'submitxml' 9. 'submitasn' 10. 'flat' 11. 'psi'

4.6.3 Returns

SearchResultBean - A SearchResultBean containing the all BIND records found, in the format specified.

4.6.4 Possible Exceptions

A BINDSOAPException is thrown if there was a problem parsing the text query provided, or if the returnType specified is unknown.

This exception will also be thrown in the instance that there was an excepted error attempting to send the results.

4.7 textSearchAttachment

4.7.1 Overview

Given a Lucene query string, returns all BIND records found, in the format specified.

4.7.2 Parameters

Name	***Type***	***Description***
Query	String	The Lucene query string.
ReturnType	String	One of the following strings: 1. 'idlist' 2. 'gipair' 3. 'go' 4. 'domain' 5. 'fasta' 6. 'cytoscape' 7. 'xml' 8. 'submitxml' 9. 'submitasn' 10. 'flat' 11. 'psi'
AttachmentType	String	One of the following strings: 1. ‘mime’ 2. ‘dime’

4.7.3 Returns

SearchResultBean - A SearchResultBean containing the all BIND records found, in the format specified.

4.7.4 Possible Exceptions

A BINDSOAPException is thrown if there was a problem parsing the text query provided, or if the returnType specified is unknown. This exception will also be thrown in the instance that there was an excepted error attempting to send the results.

5. Lucene

5.1 Brief Overview

Lucene is a high-performance, full-featured text search engine library provided by the Apache Jakarta Project <http://lucene.apache.org/java/docs/index.html>. The Lucene Java library is utilized in the BIND software package to implement text searching capabilities. Each BIND record's textual data is analyzed, and stored as a set of Fields known as a Lucene Document. This section will describe how to build complex text queries to get the most relevant data possible.

5.2 Query Syntax

As previously mentioned, Lucene is a full-featured text search engine that provides users with various operators that can be used in conjunction with search terms. In this section only a few of the operators available in Lucene will be covered, although all operators are supported by the BIND website and BINDSOAP service. For more information on the Lucene Query Syntax, please refer to the project on-line documentation <http://lucene.apache.org/java/docs/queryparsersyntax.html>.

A Lucene query is comprised of two parts: 1) the operators in the query, and 2) the terms in the query. The terms in the query can either be single terms or phrase terms (in double quotes) and can be joined together using operators.

As well, Lucene supports field specific indexing, allowing users to specify the field a term should be found in. In conjunction with the a defined data specification, field specific searches provide a powerful method for scientists and bioinformaticians to query BIND with questions ranging from broad to specific.

A field specific query has one of the following formats:

```
Field:term
Field:"phrase term"
```

Note that the field name and the query term are separated by a colon, thus making the colon an operator and a reserved character. In the case where a reserved character is part of the term to be queries, the reserved character would need to be escaped using a backslash '\'.

Other common operators available for use are Boolean operators (AND, OR, NOT) and wild card operators ('*', '?'). That said, it is important to note that Boolean operators are case sensitive. Finally, query terms can be grouped by using the open and close parenthesis.

The following example is a field specific search that contains a wild card operator:

```
pub.author.corr:jo*
```

This query will return all records in BIND that have a corresponding author whose last name begins with "JO", regardless of the case of the letters.

The next example is a Boolean operator that combines two field specific search terms:

```
taxid:9606 OR taxid: 4932
```

This query will return all records in BIND that have a molecule origin specified as *Homo sapiens* or *S. cerevisiae*.

The last example is the same as the previous example, except that we are also searching for the word "cancer" in any field. To prevent incorrect interpretation by the Lucene Query Parser, it is necessary to group the search terms using parentheses:

```
(taxid:9606 OR taxid: 4932) AND cancer
```

This query will return all records in BIND that have the word "cancer" in it and that have a molecule origin specified as *Homo sapiens* or *S. cerevisiae*.

Once again, more information on the Lucene Query Syntax can be found at <http://lucene.apache.org/java/docs/queryparsersyntax.html>.

5.3 Building Queries Using Document Field Names

As previously mentioned, a single Lucene Document contains a set of Fields corresponding to the fields in a BIND record. A Lucene Document Field is mad up of a name and a value. Here is a breakdown of some fields:

body: This is a special default field. All text in a BIND record is added to this field regardless of the type of information in the field. This field is

useful when a user querying the index without specifying a field to search in. For example, the query:

```
body: lat
```

will return back all records containing the word “lat”, regardless of which field “lat” originated from in the BIND record.

Note that the following query:

```
lat
```

Will return the same results as `body: lat`, since the body field is the default field used when no field is specified.

BINDInteraction.a.id.protein.gi: This field corresponds to the protein GenBank Identifier for molecule A and is a typical name for a field within a Lucene Document. It is derived from the BIND XML structure, which in turn is derived from the BIND ASN.1 specification <http://submit.bind.ca/asn-browser/>.

The basic rules are to take out the dashes, use dots as location steps similar to the XPath language <http://www.w3.org/TR/xpath>, and to ignore the “middle” elements. The middle elements are an artifact of converting from ASN.1 to XML. To clarify how to identify the middle elements and how to build the Lucene Field name, the following example is given:

```
<BIND-Interaction>
      <BIND-Interaction_date>
            <Date>
                  <Date_std>
                        <Date-std>
                              <Date-std_year>1999</Date-std_year>
                               ……
```

Following the above mentioned rules, a query to search for all records created within a specific year would become:

```
BINDInteraction.date.std.year: 1999
```

Notice that the XML fields `<BIND-Interaction >` , `<Date>`, `<Date-std>` are ignored when building the Lucene Field name. Thus the only fields used to build the Lucene Field name are `<BIND-Interaction_date>`, `<Date_std>` and `<Date-std_year>`.

Combining the first two fields, would give `BINDInteraction.date.std`. Note that the hyphen has been removed from `<BIND-Interaction_date>`, and that the underscores are replaced with a period. Also note that the prefix `"Date_"` from the `<Date_std>` field is removed since it is repetitive with the `"_date"` portion of the `<BIND-Interaction_date>` field.

Lastly, it is important to mention that Lucene Field names are case sensitive. Thus removing the suffix `"_date"` from `<BIND-Interaction_date>` and replacing it with the prefix `"Date_"` from `<Date_std>`, resulting in `BINDInteraction.Date.std`, will search on a field that does not exist in any Lucene document.

This methodology of building up a Lucene Field name from the structure of the XML document provides the user with the ability to specifically search on any field within a BIND record. Unfortunately, it can become quite cumbersome to build the Lucene Field name for commonly used fields. For this reason we have provided the ability to use Field name aliases (see Section 5.3 of this document).

5.4 Building Lucene Queries Using Aliased Field Names

The BINDSOAP service provides the ability for developers to use aliased field names instead of the full Lucene Document Field name when building a Lucene query. An aliased filed name can either map to multiple fields that contain similar information, or map to a single field that is commonly searched upon. Following are two examples of common aliased field names used to query BIND.

go: The aliased field `"go"` maps to multiple fields where Gene Ontology terms <http://www.geneontology.org/index.shtml> can be found. These mapped fields correspond to both fields within a BIND record and custom fields in the BIND text index that are automatically populated with GO annotation using the SeqHound API <http://blueprint.org/seqhound/apifunctslist.html>.
The “go” field is automatically converted, server side, to a Boolean query where each mapped field is OR’ed.

interaction.object.label: The aliased field `"interaction.object.label"` maps several fields where a molecule short label can be found. Included in the mapped fields are the Molecule A/B short label and Molecule A/B alias list. Once again, the

"`interaction.object.label`" field is automatically converted, server side, to a Boolean query where each mapped field is OR'ed.

Unfortunately, at the time of writing, a complete list of aliases have not been publicly published, but are available through the Field Specific Query Building (FSQB) located at http://bind.ca/Action?pg=3003.

6.0 Contact Us

To obtain more information on BIND or the BINDSOAP service, please contact us at info@bind.ca. Any comments and suggestions regarding our services should also be addressed to info@bind.ca.