DGWS Overview

Service Overview for Client Application Developers

The Discovery Gate Web Service (DGWS) enables software developers and other web services to consume Symyx data source content quickly and easily. The application programming interface (API) provides predefined methods that satisfy common use-cases.

The service has been designed to enable developers to rapidly develop client applications that can take advantage of object-oriented programming and XML. The developer uses a client-side SOAP proxy and has the freedom to choose which programming language to use.

The DGWS API supports:

- Stateless method calls for performance, ease of testing, and scalability. Stateless APIs also reduce the learning curve, accelerate application start-up, and facilitate integration into Service-Oriented Architecture (SOA). This mode is optimal for workflows that involve paging hit-sets within the limits. See "Service Request" on page 9.

- The transformation of existing stateless method calls into stateful ones with a completely transparent and time-limited session. This mode is optimal for developing high-performance asynchronous data-retrieval workflows that read large hit-sets to completion.

DGWS provides a single object model for all Symyx content and third-party data, both molecular and synthetic. To enable applications to search across a wide expanse of content that historically had no connection, Symyx created the comprehensive CompoundIndex, which de-duplicates and normalizes the data from all the molecular and reaction data sources. For example, a method returning a top-level molecule from the index has the ability to request the associated reaction information. The indexing from Molecule to Reaction is bi-directional, so applications can do reaction searching, both structural and citation-based, to return Reaction information (on top) and molecular information (underneath) from any molecule database for each molecule associated with the reaction.

To allow the developer to attenuate the retrieval of information, a set of molecule flags and reaction flags allow the user to specify areas of data of interest specific to each request. This flexibility allows control of the data retrieved by each method call, whether it involves large hierarchical datasets or specific chemical IDs. A licensing model is integrated into the retrieval flags so that search and retrieval is resolved against the entitled sources. To enable developers to write self-configuring software, we have also provide the ability to discover the licence's capabilities.

The API provides flexibility:

- XML-based software development: many types of applications require that data be both abstract and extensible. DGWS XML methods support application developers who use transforms like XSLT to convert data into a user interface without having to know details about the data.

- Object-based software development: DGWS also supports rapid application development (RAD) tools for very specific workflow applications or projects through the WSDL-based client-proxy supplied by many development environments and utilities, and supported by many programming languages, such as Java and C#. Without needing to parse or generate XML, a few lines of code, with no client-side code-base, can gain access to Symyx data from any internet connection. The service is readable and self-documenting. Many of the constructs returned can be sent in connected requests without any object reconstruction and filtration.

API Design Overview

The API is as self-documenting as possible when used from a SOAP client-proxy. When the WSDL has been consumed by a development tool, such as Eclipse Helios or Microsoft Visual Studio, the methods calls and the object-model, for both the input and output types, are somewhat self-explanatory.

All methods take a Service Request object as their first parameter. Not all of the members of this class are used by every method, but the `licenceKey` member must be populated for every request of the service.

The API provides the following kinds of methods:

- Metadata Methods
- Utility Methods
- Data Methods, which can return XML or Objects

Metadata Methods

The metadata methods return information about the service itself. The most important metadata method is `getServiceInformation`, which returns information about the service for the given licence key entitlement. This method can be used to enable an application to self-configure based on the entitlements to which the license key provides access. The metadata methods can be used by anyone with a valid licence.

Utility Methods

The utility methods, such as `convertStructureToMolfile`, support the developer but do not return any content from the underlying databases. The utility methods can be used by anyone with a valid licence.

Data Methods

The data methods are the most numerous and many are restricted to particular sets of database entitlement. Data methods return content from one or more of the underlying databases. The method name indicates what each method searches. For example, `getMoleculesByNames`.

Methods returning content as XML have the word "XML" on the end. For example, `getMoleculesByNamesXML`.

If the method ends in XML, it does not necessarily return a string as its return type. Many methods, including the ones that return XML, can return "pages" of data. Pages are returned in the form of Results containers. The containers themselves are objects, so that the developer does not need to parse the response to find out about the data contained in the response.

> One example is `DGWS.getCatalogsByIdXML`, which returns `DiscoveryGate.types.containers.XMLResults`, which is the generic container for methods returning XML.

Methods starting with `getMoleculesBy`, such as `getMoleculesByStructure`, return molecules at the top of the data-model, from the CompoundIndex for molecules. This index contains normalized and de-duplicated structures from the underlying data sources. These methods also honor the retrieval flags, which allow associated data to be returned from the underlying sources.

Methods containing the phrase `SourceMolecule`, such as `getSourceMoleculesByStructure`, specifically target the source databases. These databases sit underneath the Molecule index in the object model and supply the majority of the data.

Methods starting with `getReactionsBy`, such as `getReactionsByStructure`, return Reaction information from the Reaction databases. Reaction databases have retrieval flags too (see Reaction Retrieval Flags). However, there is no overall index for reactions because all the reaction databases conform to the same model. Specific reaction databases can be filtered out of the request by using the `sourceFilter` member of the Service Request.

Paging data

Certain methods, such as `getMoleculesByNames`, require that the data return be "paged" as a set of virtual pages.

The number of hits returned from any search is limited and therefore the page member of the `ServiceRequest` object must be used to specify the offset (starting point) and count (`number` of hits) to return. You would normally increment the offset with each paging operation. When the number of hits is less than the allowed count, it means that all the hits have been returned.

For stateless searches, which are the default, the search runs for every request regardless of whether the previous page has been requested before.

A count of zero is invalid for methods requiring page parameters to be present.

Retrieving data

Due to the size and complexity of the Object Model, it is not obvious which data is required from each method. Therefore, by default only the top-level of data is returned for a method. This means that only the basic members of the return class are populated.

To specify which area of the object model is to be retrieved from an object, specify a set of Molecule and Reaction Retrieval Flags on the `ServiceRequest` object.

Any combination of the enumerated type, `MoleculeRetrievalFlags`, can be specified on the `moleculeFlags` member array. Similarly, any combination of the enumerated type, `ReactionRetrievalFlags`, can be specified on the `reactionFlags` member array.

There are a two special retrieval flags for both the `MoleculeRetrievalFlags` and `ReactionRetrievalFlags` types:

- ID_ONLY excludes all other retrieval flags and only retrieves the molecule ID (or reaction ID) for the Molecule (or Reaction entity) and compoundId if the method targets the source molecule entities. ID_ONLY also allows the developer to exceed the normal count limits of the DataPage thereby allow more hits to found in fewer roundtrips.
- NO_STRUCTURE excludes the structure from the return data of the Molecule or Reaction entity. This flag is useful if the client already has the structure or if this data is not necessary. WAN performance is improved by using this flag because structure data can form a large part of the retrieved data.

Filtering particular sources

If more than one database can return data for a particular area or flag, your application might want to filter out a particular data source. To do this, use the `sourceFilter` of the `ServiceRequest`. Data from any `Datasource` type specified in the array will not be retrieved.

In addition, this `sourceFilter` can affect the way the search is performed. If the search knows that a particular source is not required, that source is removed from the search, thereby improving performance.

Exclusive Hits

Under normal circumstances, the selection of the Retrieval Flag does not affect the way the search is performed. The hits are normally found for the search and the related data is then serialized with the objects returned from the hits. However, if a particular area of data is the reason for the search, it is useful to return hits *solely if* such data is present. The developer can set the `exclusiveHits` member of the `ServiceRequest` to return hits if, and only if, data exists for the specified Retrieval Flags.

The flag is only effective against certain Molecule methods and does not apply to Reaction searches.

Because the ID_ONLY flag excludes all other retrieval flags, the behavior of the exclusive flag is special. By default, when these flags are used in combination, all entitled sources are required to be present in the data for the hit to found. However, the `sourceFilter` can be used to exclude sources from this requirement. This allows the developer to specify which sources must be present to return a search hit.

For performance, consider using stateful searches. With stateful searches, the query is only run once on the server, saving time.

Stateful Searches

By default, each method call to the service is stateless. Any method call can be made at any time and each call is handled separately by the service. For most use-cases, stateless methods give adequate performance. Stateless methods do not require starting and ending a session. However, because each method call is stateless, paging through large result sets require the service to re-run the query for each page offset.

If the number of hits required is large, the number of method calls can be high (even when using the ID_ONLY retrieval flag). Therefore, the search is run repeatedly to obtain the previous position in the set of hits, which is detrimental to performance.

For this situation, use the stateful mode by setting the `statefulQueryKey` member of the `ServiceRequest`. The value of this key is defined by the developer. If this key is set, the query is held open for re-use to generate the hits, which avoids the overhead of re-querying.

Limitations on stateful queries

To get details about limitations to stateful queries, use the `getServiceInformation()` method to access fields on the `ServiceInformation` class, such as:

- `maxStatefulQueries`, the number of concurrent stateful queries the service allows the users of a given license key.
- `statefulQueryTimeout`, the time limit on any stateful query. If a key is re-used outside of that timeout, the query will be re-run and the performance will be identical to the stateless mode.

If you have an open stateful query and run the same method with different parameters (or a different method), the service uses the existing parameters as stored in the stateful query rather than the new parameters.

Before you attempt to re-use a query key to start a new query, call the `closeStatefulQuery()` method with the open query key.

Glossary of Key Terms

DataPage

Generally, DGWS is stateless, and it is common that to retrieve all the hits of a query, you call a get method (such as getMoleculesByStructure) multiple times, each time with a different offset: 0, 50, 100, and so on. The DataPage object holds the offset and count for the current retrieval of hits.

DGWS classes

If you develop a client application using a tool such as Visual Studio .NET, you will see DGWS classes, such as `Availability` and `Screening`. The DGWS classes correspond to entities related to a data source. For example, `Availability` comes from the ACD data source. See the diagrams in the Retrieval Flags chapter. The API Reference refers to "objects" that are instances of DGWS classes.

WSDL

The DGWS Web Service Description Language (WSDL) enables an integrated development environment, such as Microsoft Visual Studio .NET, to provide you an XML-based remote application programming interface (API) in a programmer-friendly way. Your application needs to make a web reference to the URL with the WSDL.

Service Request

Each call to the DGWS must include your valid license key. In this sense, the service is stateless. DGWS returns up to a certain number of hits, so if your query has more than that maximum, you must re-launch the query with an offset value in the page. The offset should correspond to the maximum number of hits returned.

Currently the limits are 1000 for Retrieval Flag = ID_ONLY and 200 for all other retrievals. The limits can change without notice so Accelrys recommends that you get the limits by calling the `getServiceInformation` method.

Existence Request

In general, an application using DGWS does not need to care about which particular DGWS data source(s) the data is coming from. However, if you need to know that a particular structure is in, for example, the ACD data source, use the `ExistenceRequest` class.

Note: To discover which data sources a given structure appears in, use the getMoleculesByStructure method with the SOURCE_SUMMARY flag.

Retrieval Flag

To ensure that your search retrieves only the information that you want, DGWS has special filters known as retrieval flags. Your application can use different the retrieval flag(s) for different queries. See Retrieval Flags.

schema

The DGWS schema is the object model, and is similar to an entity-relationship diagram (ERD), except that it corresponds to the structure of the service and is not an exact representation of the underlying data sources. To learn what kinds of data are available to what kinds of DGWS classes (entities), see the Object Model chapter.

Stateful Service

DGWS provides the capability to transform existing stateless method calls into stateful ones. See Service Overview for Client Application Developers and the Tutorial.

Serialization Chain

The serialization chain is the route that DGWS uses, given its object model, to provide access to a specific type of data. The Retrieval Flags chapter has diagrams that show sections of the object model that pertain to a give type of data.

Vocabulary

A vocabulary in a DGWS data source is a controlled list of allowed terms that correspond to standardized data fields for a given data source. DGWS provides vocabularies for Classification, Procurement, Reaction, and Toxicity. For example, among the over 1200 vocabulary items in ClassificationVocabulary are `"Ace Inhibitor"`, `"AIDS Vaccine"`, and `"AMPA Receptor Antagonist"`.

See also the API Reference for `getClassificationVocabulary`, `getReactionVocabulary`, and `getToxicityVocabulary`, as well as the Tutorial on Retrieving and Caching Vocabularies (C#) or Retrieving and Caching Vocabularies (Java).

URLs

DiscoveryGate Web Service (DGWS) home page

`https://www.discoverygate.com/webservice/1.2/`

WSDL for the Discovery Gate Web Service

`https://www.discoverygate.com/webservice/1.2/DGWS?WSDL`

schema for the Discovery Gate Web Service

The 1.2 version has two schemas:

`https://www.discoverygate.com/webservice/1.2/schema/DGWS-1.2.700.xsd`

for the main XML schema and

`https://www.discoverygate.com/webservice/1.2/schema/DGWS-types-1.2.700.xsd`

for the XML schema that is being included into the main schema file.

Developer's Guide

`https://www.discoverygate.com/webservice/1.2/docs/devguide/index.html`

which is also available in PDF as

`https://www.discoverygate.com/webservice/1.2/docs/devguide/dgws-devguide.pdf`

The API reference

`https://www.discoverygate.com/webservice/1.2/docs/api-reference/index.html`

Product marketing home page for DGWS

`http://www.symyx.com/products/software/database-access/web-services/index.jsp`

C# and Java

The source code underlying DGWS is written in Java. To write a client application in a different language, such as C#.NET, you must reinterpret some Java-centric aspects of this API Reference. For example, .NET uses Properties where Java uses get and set methods.

Retrieval Flags

Overview

DGWS provides both Molecule Retrieval Flags and Reaction Retrieval Flags. Molecule and reactions flags control the data that the DiscoveryGate Web Service (DGWS) returns in Molecule and Reaction records. To improve performance and bandwidth usage by downloading only relevant data, use the retrieval flags.

Note: To learn how to set retrieval flags in a `ServiceRequest`, see the API Reference.

Stateless service

Generally speaking, DGWS is a stateless service. Except for molecule ID_ONLY and reaction ID_ONLY, which can return up to 5000 hits, all the other flags return at most 50 hits. Therefore, to get the next 50 hits, set a value for the offset and re-execute the API method. To collect all the possible hits, each method call requires that your code increment the offset until less than 50 hits are returned.

Combining molecule flags

You can combine certain flags together. For example, if you want

```
mol = GetMolecule(GetServiceRequest(MoleculeRetrievalFlags.TOXIC_EFFECT,
MoleculeRetrievalFlags.LITERATURE), molName, molFormula);
```

Data source level and the "top-level" index

DGWS has a "top-level" index that is hierarchically above the data from the data source level, that is, data from any of the DGWS data sources (ACD, SCD, TOX, MDDR, and so on). The methods in the API allow your application to retrieve data from both levels without you having to consider which data source the data is from.

You can add properties (calculated properties, such as molweight) to the top-level by using the SOURCE_PROPERTIES flag. Likewise you can add the structure at the top-level with the SOURCE_STRUCTURE flag. You can also see the top-level structures and Ids collected into a single hierarchy by using the SOURCE_SUMMARY flag that lists the datasources that have the structure you are searching for.

sourceFilter

You also have the option to consider specific data sources and return data from both the top-level and the data source level. For example, the `Availability` DGWS class comes from the ACD data source. The `Screening` DGWS class comes from the SCD data source. Both `Availability` and `Screening` are returned with the PROCUREMENT_SUPPLIER or PROCUREMENT_PRICING flag. If you have access to both data sources, but do not want data from one of these data sources, filter-out that data source by using the `sourceFilter` of the `ServiceRequest` object. The `sourceFilter` can also be used to filter out the combined sources for a number of retrieval flags and also Reactions (see the API Reference).

You can also filter out Names and Properties from the data sources you have access to. In this case, properties are from the data source-level because the structure can differ at this level from the top-level index.

Note: It is not possible to search computed structure properties.

Combining molecule and reaction flags

Molecule to reaction

In certain cases, it is possible to combine a molecule retrieval flag and a reaction retrieval flag. For example, you can use the REACTION_PRODUCT flag on a Molecule to get a Reaction. From that Reaction, you can use a reaction retrieval flag, such as CITATION.

Reaction to molecule

Similarly, you can use the MOLECULE flag on a Reaction to get a Molecule. From that Molecule, you can use a molecule retrieval flag, such as PROCUREMENT_SUPPLIER.

Note: An attempt to go from Reaction to Molecule, then back to Reaction would cause an infinite loop and is not allowed. Therefore, in this example, once you get from Reaction to Molecule, an exception is thrown if you try to use a reaction flag such as REACTION_PRODUCT.

Flags and search hits

In most cases, setting a flag can change the *number* of search hits that DGWS returns. For example, an exact search with no flags on the Molecule known as acetylsalicylic acid returns one (1) top-level (DGWS index) hit that includes the following top-level data: `id`, `Structure`, `datasource`, `molformula`, `molweight`, `nemakey`, `nemakeystatus`, and `stmoddate`. This search returns zero (0) lower-level hits from any external data source.

Adding the `DRUG` flag retrieves one (1) lower-level hit from the MDDR external data source, which contains additional data for `companies`, `licenses`, `sources`, and `trademarks`.

However, the following flags do not change the *number* of hits. Rather, they change the *type* of data returned: ID_ONLY, NO_STRUCTURE, SOURCE_PROPERTIES, SOURCE_STRUCTURE. In the case of acetylsalicylic acid, the hit count remains one (1) top-level (DGWS index) hit and no lower-level (external data source) data. For example, setting `ID_ONLY` causes DGWS to exclude all the top-level data except the `id`.

Molecule Retrieval Flags

Molecule retrieval flags also apply both to methods that retrieve molecules and to methods that retrieve reactions at the top level, when Molecules (such as reactants) are retrieved at a lower level by specifying `ReactionRetrievalFlags.MOLECULE`

Here is an example that specifies a particular retrieval flag, IDENTIFICATION:

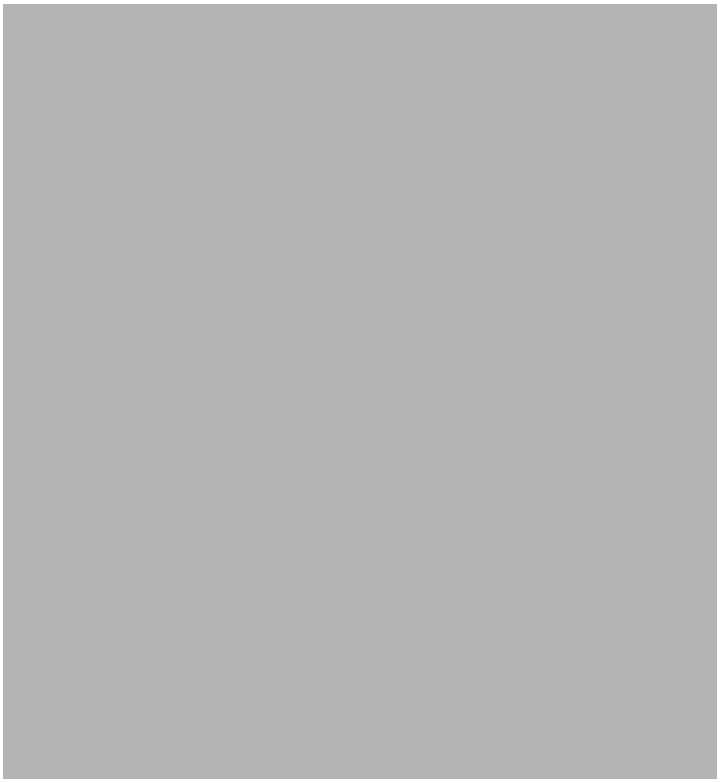
```
MoleculeResults mr =
rds.getMoleculesByStructure(GetServiceRequest(MoleculeRetrievalFlags.IDENTIFICATION),
myMoleculeSearchArray);
```

CARCINOGENIC

Use case: to get carcinogenic data, such as lethal concentration, for a given chemical substance.

CLASSIFICATION

Use case: To get data such as the drug activity (such as Analgesic, Antirheumatic) of a given chemical substance.



DRUG

Use case: To get drug-related data for a given chemical substance: `companies`, `licenses`, `sources`, and `trademarks`.

ID_ONLY

Use case: To get only the ID number of the molecule. See Flags and search hits.

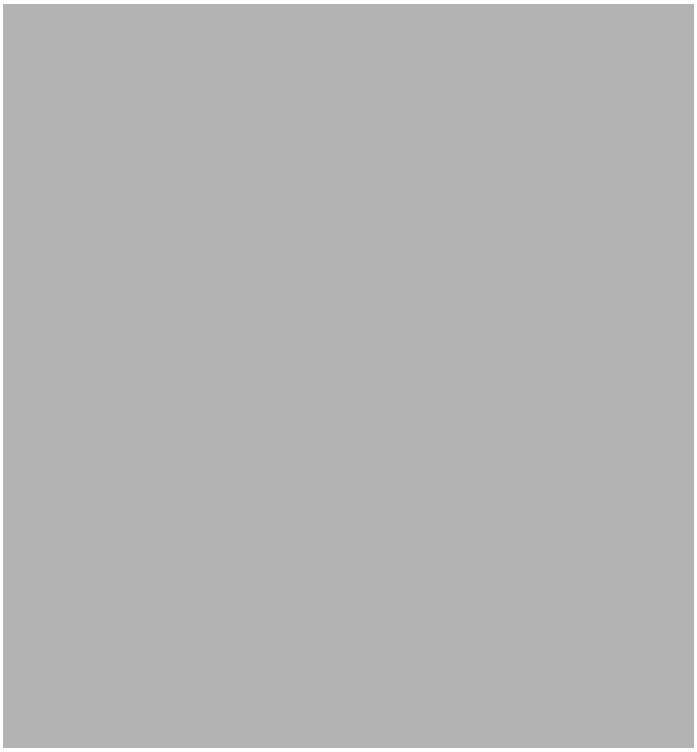
Note: This flag is stateful, so you do not need to (re)set an offset to get more than 50 hits. However, the maximum number of hits is 5000.

IDENTIFICATION

Use case: To retrieve data in the names and external arrays of Molecules returned from DGWS. If this flag is not present, `names` will be null.

LITERATURE

Use case: To get literature citations for a drug.

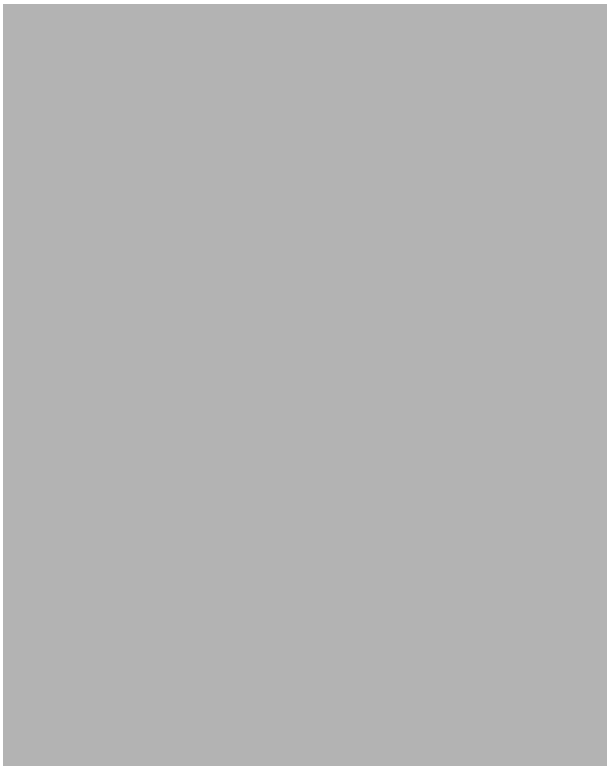


NO_STRUCTURE

Use case: To improve performance by omitting the transmission of the structure molfile string, which can be useful if there is no need to render the molecule. See Flags and search hits.

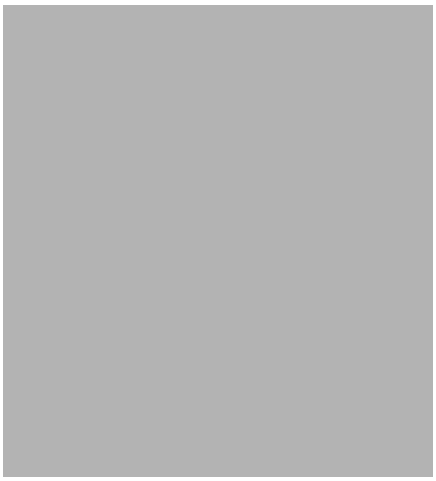
PROCUREMENT_PRICING

Use case: To get product data with the sublevel of Package (which has pricing information), but without the sublevel of ProductData, which is available by using PROCUREMENT_PROPERTIES.



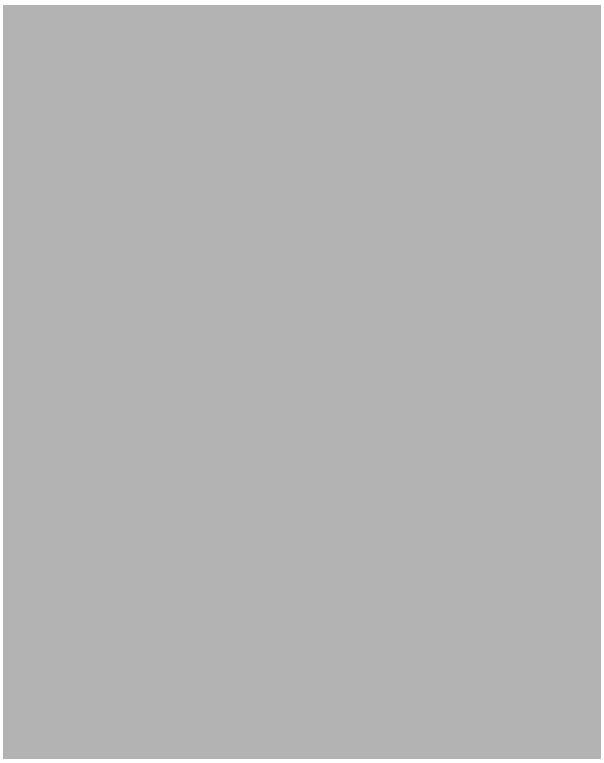
PROCUREMENT_PRODUCT

Use case: To get product data without the sublevels of ProductData and Pricing, which are available with PROCUREMENT_PROPERTIES (ProductData) and PROCUREMENT_PRICING (Package).



PROCUREMENT_PROPERTIES

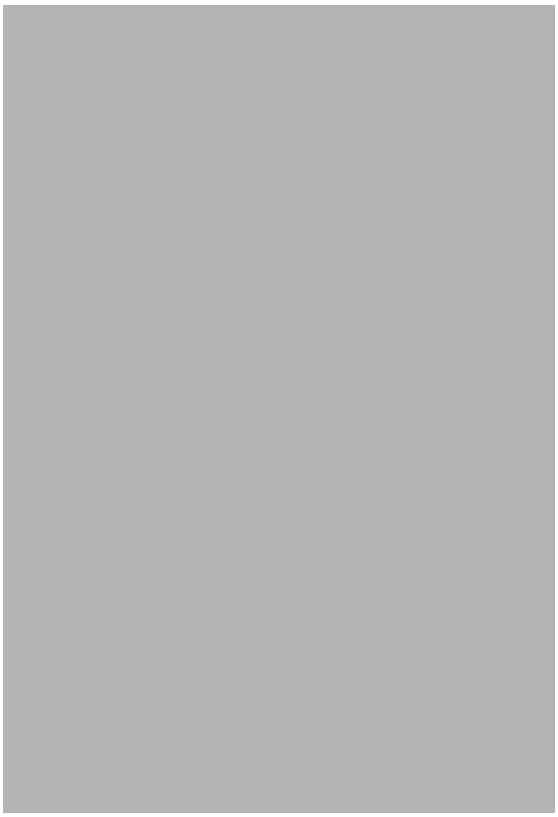
Use case: To get product data with the sublevel of ProductData but without the sublevel of Package, which is available by using PROCUREMENT_PRICING.



PROCUREMENT_SUPPLIER

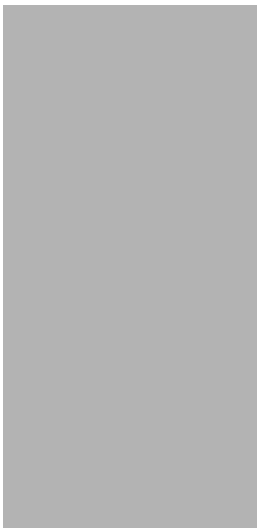
Use case: To get a list of the companies that supply a specific chemical substance.

Note: Distributor and Catalog, although related, are not serialized by this flag.



PROPERTIES

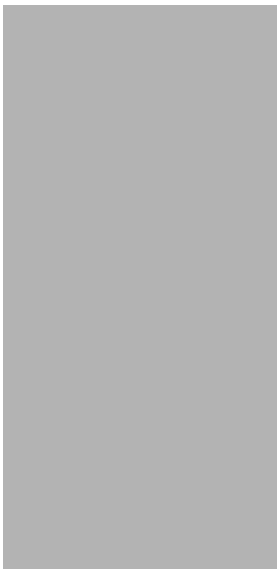
Use case: To get the chemical properties (such as HDONORS and CLOGP) for a specific chemical substance from the data-source level.



REACTION_PRODUCT

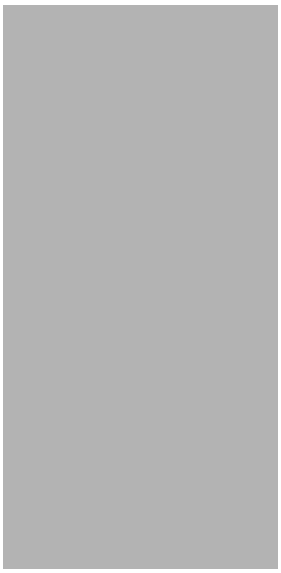
Use case: To get a a reaction that yields a specific molecule as a product.

See Combining molecule and reaction flags.



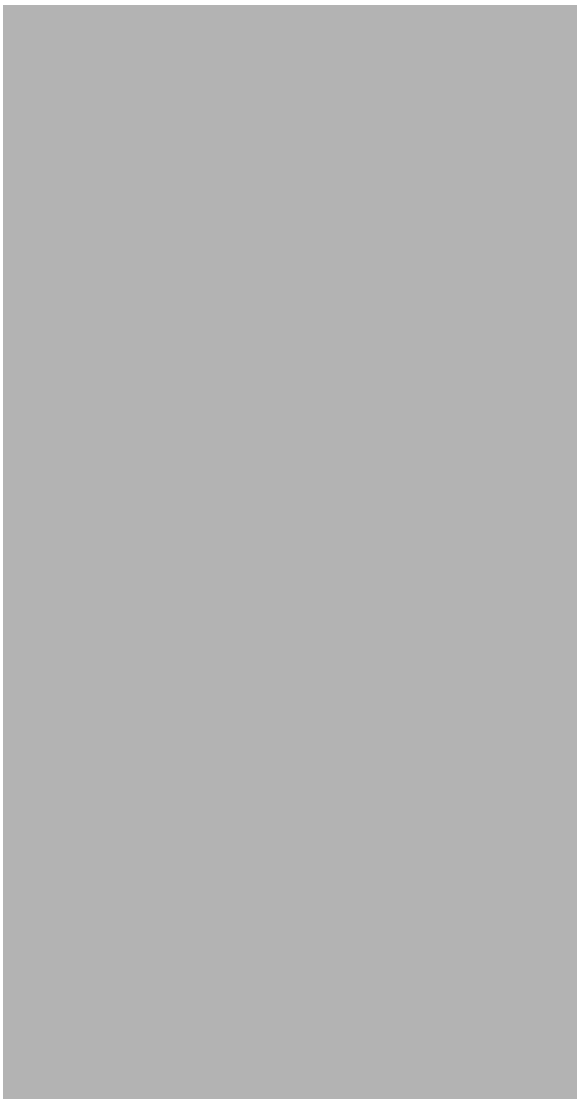
REACTION_REACTANT

Use case: To get a a reaction that uses a specific molecule as a reactant.



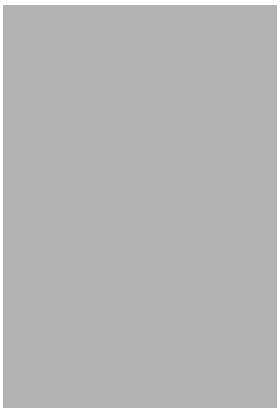
SOURCE_PROPERTIES

Use case: To get only the "top-level" data from the DGWS index instead of getting data from the data source level. See Flags and search hits.



SOURCE_STRUCTURE

Use case: To get the molfile string from the data source . See Flags and search hits.



SOURCE_SUMMARY

Returns the `datasource` list for a given structure, along with that structure's `compoundId` for each data source. Data sources includes both Discovery Gate source databases (such as ACD, TOX, MDDR) as well any external data sources (such as PUBCHEM) that also contain the structure.

The listing of structure Ids in external sourcess is limited to structures that are also in one or more DGWS data sources, and the structure itself is not returned. However, if SOURCE_SUMMARY flag is used in combination with SOURCE_STRUCTURE flag, then DGWS also returns the molfile string of that structure from each of the data sources.

TOXIC_CHEMICAL

Use case: To get a list of initiators and promoters related to the toxicity of a molecule.

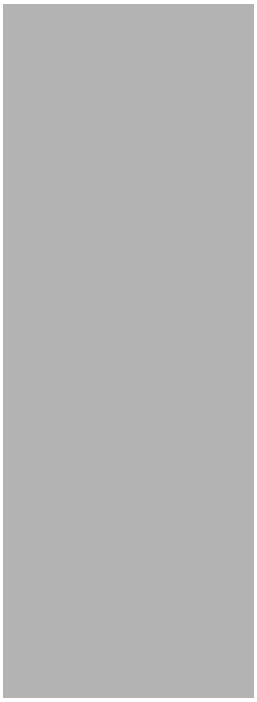
TOXIC_EFFECT

Use case: To get data about the toxic effects and lesions associated with a specific molecule.

Reaction Retrieval Flags

CITATION

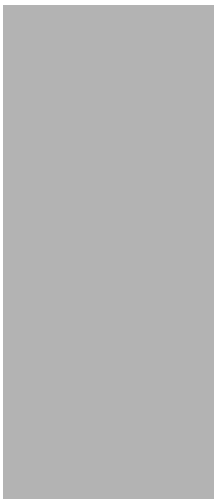
Use case: to get data about literature citations for a specific reaction, such as `author`, `journal Coden`, `journal Pg`.



DISCRETE_CLASS_CODES

Use case: to get the classification codes of a given reaction:

- `broad` (atoms of the reaction center)
- `medium` (includes the alpha atoms)
- `narrow` (alpha and beta atoms).



ID_ONLY

Use case: To get only the ID number (`rxnmdlnumber`) of the reaction.

Note: This flag is stateful, so you do not need to (re)set an offset to get more than 50 hits. However, the maximum number of hits is 5000.

MOLECULE

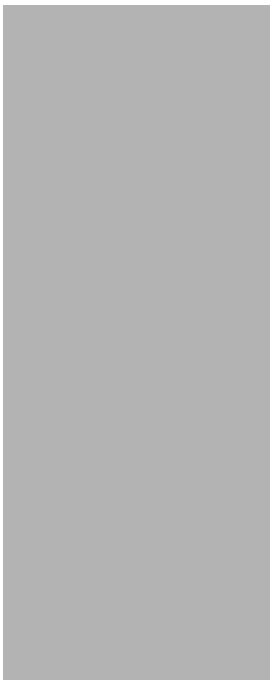
Use case: For a given reaction, to get molecule data about a reactant, product, or catalyst. See Flags and search hits as well as REACTION_PRODUCT and REACTION_REACTANT.

NO_STRUCTURE

Use case: To improve performance by omitting the transmission of the structure rxnfile string, which can be useful if if there is no need to render the reaction. For a reaction, the top-level data includes the `rxnmdlnumber`, `datasource`, and `max Overall Yield`. See Flags and search hits.

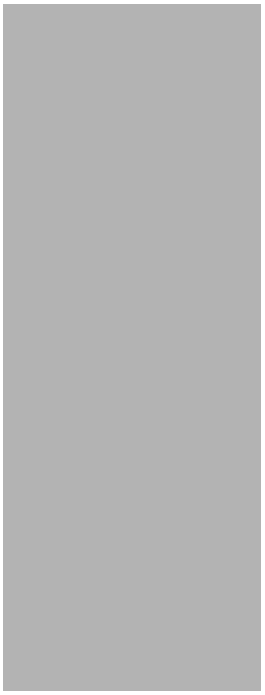
PATHS_AND_STEPS

Use case: to get data for each `path` and `step` (including `SUMMARY`) of a reaction.



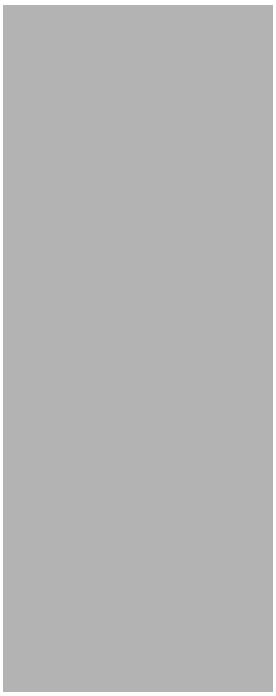
SCHEME

Use case: To get a reaction scheme. A portion of a scheme is shown below.



STEP_CONDITIONS

Use case: To get the conditions of the steps in a reaction scheme, such as `ph Max`, `stepno`, and `time Max`.



Object Model

Overview

Some Integrated Development Environments (IDEs), such as Eclipse Helio, Oracle JDeveloper, or Microsoft Visual Studio, can show the object model of the Discovery Gate Web Service as a diagram. The object model diagram is similar to an entity-relationship diagram about the data sources your client application can access through the Discovery Gate Web Service. To work with the diagram:

1. Open `Molecule.xsd` in the IDE tool. This file is named `Molecule.xsd` because it contains the metadata of the `Molecule` object.
2. Expand any DGWS class (entity) you want.

Limitation

For a small number of DGWS classes, the IDE might allow the data to be shown recursively. Although in such cases, the graphical representation appears as a sort of infinite loop, the actual dynamic serialization of the DGWS service avoids the issue.

Note: The relationships between Molecule, on the one hand, and Company and Product, on the other, are actually mediated by the Availability and Screening classes, which are subsumed under Molecule.

DGWS Tutorial for Java Client

This tutorial explains the key concepts that are illustrated in its sample applications:

Running Tutorial Code

The following tutorial has been developed using the popular, free Eclipse Java IDE (Helios) which can be downloaded from `http://www.eclipse.org/downloads/`

For this tutorial you will need the "Eclipse IDE for Java EE Developers" for your specific platform. However, the tutorial should also work with an earlier version of Eclipse.

Prerequisites

We assume that Eclipse and a Java 1.6 SDK has been installed, and that the user knows how to use Eclipse and write code in Java.

First Steps Accessing the Web Service

Step 1: Creating an Eclipse Project for the DGWS Java Client

1. Start your Eclipse IDE and create a new project called DGWSJavaTutorial.
2. You should have a new Java project with one folder called 'src'. Right-click the src folder and select "New >" and then "Other..." (or just type [CTRL]+[N]).
   a. In the dialog that displays, scroll down to the section "Web Services" and select "Web Service Client". Click "Next >"
   b. In the "Service definition:" box, type the URL to the DGWS WSDL: https://www.discoverygate.com/webservice/1.2/DGWS?wsdl
   c. In the "Client type:" drop-down, keep the option "Java Proxy".
   d. Move the slider down until it says "Assemble client".
   e. Make sure that the box "Monitor the Web service" is unchecked.
   f. Click "Next >" and leave the suggested "Output folder" unchanged.

g. Click "Finish"

h. Wait until the following two new Java packages are created under the 'src' folder"

- `com.discoverygate.webservice`
- `com.discoverygate.webservice.types`

Take some time to browse through the generated Java classes, especially the ones in the types package.

We are now ready to build our first tutorial client.

Step 2: Creating a Service Request and Getting Service Information

1. Right mouse-click on the 'src' folder again and select "New >" ? "Class"
2. Enter com.discoverygate.tutorial as Package and Example1 as Name. Make sure to check the box "public static void main(String[] args)" to generate the main method.
3. Click "Finish"

You should now have a skeleton class file, which we will use to write our first example. Modify the class skeleton as shown below:

Example 1 - basic service request

```
package com.discoverygate.tutorial;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.discoverygate.webservice.DGWS;
import com.discoverygate.webservice.DiscoveryGate;
import com.discoverygate.webservice.DiscoveryGateLocator;
import com.discoverygate.webservice.types.DataPage;
import com.discoverygate.webservice.types.ServiceInformation;
import com.discoverygate.webservice.types.ServiceRequest;

public class Example1 {

  private static final  String LICENSE_KEY = "<Your license key goes here>";

  /*
   * This method will create the minimal ServiceRequest object.
   */
  private static ServiceRequest createSr() {
    ServiceRequest sr = new ServiceRequest();
    sr.setLicenseKey(LICENSE_KEY);
    DataPage dp = new DataPage();
    dp.setOffset(0);
    dp.setCount(10);
    sr.setPage(dp);
    sr.setStatefulQueryKey("");
    return sr;
  }

  /**
   * @param args
   */
  public static void main(String[] args) {
    DiscoveryGate dg = new DiscoveryGateLocator();
    try {
      DGWS service = dg.getDGWS();
      ServiceInformation si = service.getServiceInformation(createSr());
      System.out.println("Service Version: " + si.getVersion());
      System.out.println("Copyright: " + si.getCopyright());
    } catch (RemoteException e) {
      e.printStackTrace();
    } catch (ServiceException e) {
      e.printStackTrace();
    }
  }

}
```

To execute the example class in Eclipse:

In the Eclipse "Project Explorer" select your newly create class "Example1.java" and right-click. From the pop-up menu, select "Run As >" and then "Java Application". This executes the "main()" method of the currently selected Java class file.

The output is printed into the "Console" windows, which is usually in a separate pane at the bottom of your Eclipse window.

NOTE: When running your examples inside Eclipse, you might see a Warning in the console similar to the following:

```
Nov 17, 2010 12:18:44 PM org.apache.axis.utils.JavaUtils isAttachmentSupported
WARNING: Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.
```

We can safely ignore this warning because we did not add a library to the classpath that the web services framework Apache Axis depends on. A more detailed description of this issue and a potential solution can be found at `http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.wst.doc.user/topics/limitations.html`

Step 3: Using a Method that Returns a Structure

The following example makes a call that returns the structure for "Aspirin".

Similar to the example above, we create a new class inside the same `com.discoverygate.tutorial package`, but this time call it `Example2`.

Example 2 - get a structure

```
package com.discoverygate.tutorial;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.discoverygate.webservice.DGWS;
import com.discoverygate.webservice.DiscoveryGate;
import com.discoverygate.webservice.DiscoveryGateLocator;
import com.discoverygate.webservice.types.DataPage;
import com.discoverygate.webservice.types.Datasource;
import com.discoverygate.webservice.types.Molecule;
import com.discoverygate.webservice.types.MoleculeResults;
import com.discoverygate.webservice.types.MoleculeRetrievalFlags;
import com.discoverygate.webservice.types.ServiceRequest;

public class Example2 {

  private static final  String LICENSE_KEY = "<Your license key goes here>";

  /*
   * This method will create the minimal ServiceRequest object.
   */
  private static ServiceRequest createSr() {
    ServiceRequest sr = new ServiceRequest();
    sr.setLicenseKey(LICENSE_KEY);
    DataPage dp = new DataPage();
    dp.setOffset(0);
    dp.setCount(10);
    sr.setPage(dp);
    sr.setStatefulQueryKey("");
    MoleculeRetrievalFlags[] mrf = new MoleculeRetrievalFlags[]{};
    sr.setMoleculeFlags(mrf);
    Datasource[] sourceFilter = new Datasource[]{Datasource.CINDEX};
    sr.setSourceFilter(sourceFilter);
    return sr;
  }

  /**
   * @param args
   */
  public static void main(String[] args) {
    DiscoveryGate dg = new DiscoveryGateLocator();
    try {
      DGWS service = dg.getDGWS();
      String[] names = new String[]{"Aspirin"};

      MoleculeResults mr = service.getMoleculesByNames(createSr(), names);
      System.out.println("We got " + mr.getContainedCount() + " result(s).");
      System.out.println("Query count is " + mr.getQueryCount());
      Molecule[] molecules = mr.getRecords();
      System.out.println("The structure for the first found molecule is (Mol File):");
      System.out.println(molecules[0].getStructure());
    } catch (RemoteException e) {
      e.printStackTrace();
    } catch (ServiceException e) {
      e.printStackTrace();
    }
  }

}
```

Step 4: Searching for Molecules and Reactions

Example 3 - One structure as input for SSS search

Suppose you do a search with one element in the MoleculeSearch array that has sodium acetylsalicylate as the input structure and `MoleculeSearchType.SUBSTRUCTURE` as the type. This returns any molecule for which Sodium Acetylsalicylate is a substructure.

```
package com.discoverygate.tutorial;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.discoverygate.webservice.DGWS;
import com.discoverygate.webservice.DiscoveryGate;
import com.discoverygate.webservice.DiscoveryGateLocator;
import com.discoverygate.webservice.types.DataPage;
import com.discoverygate.webservice.types.Datasource;
import com.discoverygate.webservice.types.Molecule;
import com.discoverygate.webservice.types.MoleculeResults;
import com.discoverygate.webservice.types.MoleculeRetrievalFlags;
import com.discoverygate.webservice.types.MoleculeSearch;
import com.discoverygate.webservice.types.MoleculeSearchType;
import com.discoverygate.webservice.types.Parenthesis;
import com.discoverygate.webservice.types.SearchOperator;
import com.discoverygate.webservice.types.ServiceRequest;

public class Example3 {
  private static final String LICENSE_KEY = "77B9858A9E8D6F7BE0401EAC16FD2EA6";
  // The following represents Sodium-qacetylsalicylate
  private static final String MOL_CHIME_STRING =
"CYAAFQwAYewQFPfJfx616ZLb6rNugDNdp8dA8cRIkKkpZ55HBpDGtTgMMRQ$XWYlh9ME^V9Z6IcS98biuRVPy8whVCD
x5e6NILuAen5LsA$j1$6Gaty9$tJzeoTfv$o3GKH8tnf^NkJ^RCJYpzLUw2SgFDaACHDiRBxk4MgoaKDVGYlPxcfvpQ4
aGC7Sph4SOgCzvxaR0g1olMiK2EqiyHkiKYgaN51qxOXSXrZeYQSdi$smMJDSxNy1lnA4iWYsbgF6oL3OMVZj2gSC05N
NqWjqGXyIRyN0Y638ezmbK8luWXxoEMrto5FNnfB3x1kY4M1^nQUVlV4I7$iudF^Q^YNfeQLlPKuUUWqKSq2Z11Ob71Z
RMnUfSwVvy92cm0aaqRdKSKuk0TryKb$NuSymq58OWEf1^1HY5w7Vwocufewm7e7MN8fTeTvjiA";

  /*
   * This method will create the minimal ServiceRequest object.
   */
  private static ServiceRequest createSr() {
    ServiceRequest sr = new ServiceRequest();
    sr.setLicenseKey(LICENSE_KEY);
    DataPage dp = new DataPage();
    dp.setOffset(0);
    dp.setCount(10);
    sr.setPage(dp);
    sr.setStatefulQueryKey("");
    MoleculeRetrievalFlags[] mrf = new MoleculeRetrievalFlags[]{};
    sr.setMoleculeFlags(mrf);
    Datasource[] sourceFilter = new Datasource[]{Datasource.CINDEX};
    sr.setSourceFilter(sourceFilter);
    return sr;
  }
```



```
  /**
   * @param args
   */
  public static void main(String[] args) {
    DiscoveryGate dg = new DiscoveryGateLocator();
    try {
      DGWS service = dg.getDGWS();

      MoleculeSearch[] molSearches = new MoleculeSearch[1];
      MoleculeSearch moleculeSearch = new MoleculeSearch();
      moleculeSearch.setStructure(MOL_CHIME_STRING);
      moleculeSearch.setSearchType(MoleculeSearchType.SUBSTRUCTURE);
      moleculeSearch.setBooleanOperator(SearchOperator.NONE);
      moleculeSearch.setParenthesis(Parenthesis.NONE);
      moleculeSearch.setCustomFlags("");
      molSearches[0] = moleculeSearch;
      MoleculeResults mr = service.getMoleculesByStructure(createSr(), molSearches);
      System.out.println("We got " + mr.getContainedCount() + " result(s).");
      System.out.println("Query count is " + mr.getQueryCount());
      Molecule[] molecules = mr.getRecords();
      System.out.println("The structure for the first found molecule is (Mol File):");
      System.out.println(molecules[0].getStructure());
    } catch (RemoteException e) {
      e.printStackTrace();
    } catch (ServiceException e) {
      e.printStackTrace();
    }
  }
}
```

Note: If the 'not' member of the MoleculeSearch element is set to true (moleculeSearch.setNot(true)), the search returns any result molecule for which the input structure is NOT a substructure.

Understanding offsets

In the example above, the printout for the number of results is 10 while "Query count" shows -1 (in the previous example the two numbers were the same). This is an important concept to understand.

The property `containedCount` always represents the actual number returned in this specific result. Since we set our page size to 10, `containedCount` will be that same number as soon as the total number of results is larger than our page size. Such a case will be indicated by setting `queryCount` equals to -1, telling us that there are more records to be retrieved.

So, how can we retrieve the additional records? The answer is by changing the *offset value* to a value such as 10 in the `DataPage` for our current `ServiceRequest`, and then making the same service call again. If this subsequent call will then return a `queryCount` of -1 again, we will have to increment the offset again by the count value until our queryCount will have a value <> -1 and showing the total number or results returned in this search.

Complex searches with nested parentheses

If your search criteria involve four or more structures with a mixture of ANDs and ORs, the logic might require nested or double parentheses. For example, the following are different:

```
sss[struct1] AND (sss[struct2] OR sss[struct3] OR sss[struct4])
(sss[struct1] AND (sss[struct2] OR sss[struct3])) OR sss[struct4]
```

For the second case, indicate that the closing of struct3 is double parenthesis:

```
))
```

that is, a parenthesis with a `parenthesisCount` of 2.

Example 4 - complex search with parentheses

```
package com.discoverygate.tutorial;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.discoverygate.webservice.DGWS;
import com.discoverygate.webservice.DiscoveryGate;
import com.discoverygate.webservice.DiscoveryGateLocator;
import com.discoverygate.webservice.types.DataPage;
import com.discoverygate.webservice.types.Datasource;
import com.discoverygate.webservice.types.Molecule;
import com.discoverygate.webservice.types.MoleculeResults;
import com.discoverygate.webservice.types.MoleculeRetrievalFlags;
import com.discoverygate.webservice.types.MoleculeSearch;
import com.discoverygate.webservice.types.MoleculeSearchType;
import com.discoverygate.webservice.types.Parenthesis;
import com.discoverygate.webservice.types.SearchOperator;
import com.discoverygate.webservice.types.ServiceRequest;

public class Example4 {

  private static final String LICENSE_KEY = "<Your license key goes here>";
  // The following represents Sodium-qacetylsalicylate
  private static final String MOL_CHIME_STRING =
"7bwCwtwA3FwQEkegs76onElRauPtmkwOGQ7^mdlmIQfUAN2SFVD64HKwz3SfqslhcKMzu1HnLViTvctuzbOh9GhLZwD
tS4ZFMGHO0M7vQx68FOfgxq7n70tyh1Bvfw9cHa5v4cS7USCZgZA6Aw$DkLG6VFOnojMiVGAADrNnGDD6GtImCmFaDxV
BMeMnEDlEK6zxyc6R4IbU4Mq6ujKpVHGR2kymm$1FHtVe7nqgkooLGl$p85EJpnnRGgSoO6s1nbkASAVpd$kiqp9XEpe
n1lTyiRxd30EPd^2dRX4Lfv2C6KRXtTDK5iwS^TYpVwWKEhcOfnTY2W^l$7q$U8j1p$wWVeU8zosWfTS1VNkeY595vM3
EcT81t5yaSNZVUq5qMl6LG79lNhH3jbr4dYjosMwPr6hnHTytlNvAzXC8O4FsX943pyN";

  /*
   * This method will create the minimal ServiceRequest object.
   */
  private static ServiceRequest createSr() {
    ServiceRequest sr = new ServiceRequest();
    sr.setLicenseKey(LICENSE_KEY);
    DataPage dp = new DataPage();
    dp.setOffset(0);
    dp.setCount(10);
    sr.setPage(dp);
    sr.setStatefulQueryKey("");
    MoleculeRetrievalFlags[] mrf = new
MoleculeRetrievalFlags[]{MoleculeRetrievalFlags.PROCUREMENT_PRICING};
    sr.setMoleculeFlags(mrf);
    Datasource[] sourceFilter = new Datasource[]{Datasource.CINDEX};
    sr.setSourceFilter(sourceFilter);
    return sr;
  }

  /**
   * @param args
   */
  public static void main(String[] args) {
    DiscoveryGate dg = new DiscoveryGateLocator();
    try {
      DGWS service = dg.getDGWS();

          ////////////////////////////////////////////////////////////////////
          // Four structures as input, reflecting the following logic:
          // (sss[struct1] AND (sss[struct2] OR sss[struct3])) OR sss[struct4]
```

```
        ////////////////////////////////////////////////////////////////////

        MoleculeSearch[] ms = new MoleculeSearch[4];
        ms[0] = new MoleculeSearch();

        final String sodiumAcetylsalicylateChimeString =
"7bwCwtwA3FwQEkegs76onElRauPtmkwOGQ7^mdlmIQfUAN2SFVD64HKwz3SfqslhcKMzu1HnLViTvctuzbOh9GhLZwD
tS4ZFMGHO0M7vQx68FOfgxq7n70tyh1Bvfw9cHa5v4cS7USCZgZA6Aw$DkLG6VFOnojMiVGAADrNnGDD6GtImCmFaDxV
BMeMnEDlEK6zxyc6R4IbU4Mq6ujKpVHGR2kymm$1FHtVe7nqgkooLGl$p85EJpnnRGgSoO6s1nbkASAVpd$kiqp9XEpe
n1lTyiRxd30EPd^2dRX4Lfv2C6KRXtTDK5iwS^TYpVwWKEhcOfnTY2W^l$7q$U8j1p$wWVeU8zosWfTS1VNkeY595vM3
EcT81t5yaSNZVUq5qMl6LG79lNhH3jbr4dYjosMwPr6hnHTytlNvAzXC8O4FsX943pyN";

        ms[0] = new MoleculeSearch();
        ms[0].setParenthesis(Parenthesis.OPEN);
        ms[0].setSearchType(MoleculeSearchType.SUBSTRUCTURE);
        ms[0].setStructure(sodiumAcetylsalicylateChimeString);
        ms[0].setBooleanOperator(SearchOperator.AND);
        ms[0].setCustomFlags("");

        final String benzeneChimeString =
"OYwVwf1AyBwQ3gtkYcHMxn6SfUljcwzqzPlD8D$XQvR^sYiBrCgtPNrQn$rheTgQlzT6G^RidoIhHJEK^^h0mkXZnNv
AvRh0qQlnKOpNzz5hHoZK0ievzAktHdIX3db3WsEERvXLD8hLIY0vM6UADoOiPv1kFoGVLlMZqDNrFIULTDxvQDFWLho
iNg4owcDG4ul2FoAZXGn5l$vgl8NvEY6TD5PdBWUcCQRKnhJ4SokqVx0879VKrJXV^t4r1nOZJC84YDgesveHkpcBKpG
";

        ms[1] = new MoleculeSearch();
        ms[1].setParenthesis(Parenthesis.OPEN);
        ms[1].setSearchType(MoleculeSearchType.SUBSTRUCTURE);
        ms[1].setStructure(benzeneChimeString);
        ms[1].setBooleanOperator(SearchOperator.OR);
        ms[1].setCustomFlags("");

        final String sodiumIonChimeString =
"8YASIz5ALAwQ7O7uORgiVrMfibNYyqfBA3mHLTNbBF5GJ3bLQnBsNF0Zk8QShFlQ5FILDpPFpH9L2uQoQRlbKaDK8KI
eKK0PAySqoczsxYjEcUohW23MTCBMpYu9XgfA4q0U7PDKJS4Hue6UvHlog76ITNgn2PQXZs8pwBszb6qOd232DT5wc3y
BrSC";

        ms[2] = new MoleculeSearch();
        ms[2].setSearchType(MoleculeSearchType.SUBSTRUCTURE);
        ms[2].setStructure(sodiumIonChimeString);
        ms[2].setParenthesis(Parenthesis.CLOSED);
        ms[2].setParenthesisCount(2);
        ms[2].setBooleanOperator(SearchOperator.OR);
        ms[2].setCustomFlags("");
        final String napthaleneChimeString =
"JYAu$j7AlAwQBfHAybSPhbKFN35N0aQLH2W6tjPQgXdVX6$^EVjYXsLoo^4EXRQSo5gcM5yG5V7UsxZaAavdCeJ7zBR
MCit4FMxGGGw9b97vmybFNGK7v9YyArL8vwLvpP9neEA8dlQGwMA9A9QGV5wvHdRhHaGRIEW0r8i6pgswIoA2wqUrU87
gjd0PJRRYC^cuvVImixhldW4tuovp1$vw05Ll4J7Wcolvz$eiHoAEQbZqdE7ZrjpnqLXjtiztYXe36PyfaLt^b3V18qM
5y3iSl6VZWmsFshStivYW13olNZJwXazW1WGtWrBFLRt^lWWosbNFjD8E8Rz8OxWAyVK";

        ms[3] = new MoleculeSearch();
        ms[3].setParenthesis(Parenthesis.NONE);
        ms[3].setSearchType(MoleculeSearchType.SUBSTRUCTURE);
        ms[3].setStructure(napthaleneChimeString);
        ms[3].setBooleanOperator(SearchOperator.NONE);
        ms[3].setCustomFlags("");

    MoleculeResults mr = service.getMoleculesByStructure(createSr(), ms);
    System.out.println("We got " + mr.getContainedCount() + " result(s).");
    System.out.println("Query count is " + mr.getQueryCount());
```



```
      Molecule[] molecules = mr.getRecords();
      for(int i = 0; i < mr.getContainedCount(); i++) {
        Molecule m = molecules[i];
        System.out.println("Molecule #" + (i+1) + ":");
        System.out.println("Mol formula: " + m.getMolformula());
        System.out.println("Mol weight:  " + m.getMolweight());
      }
    } catch (RemoteException e) {
      e.printStackTrace();
    } catch (ServiceException e) {
      e.printStackTrace();
    }
  }

}
```

Step 5 : Converting Structure Types

The convertStructureToMolfile method supports converting an array of structures to the molfile format. The source array can be in the SMILES format or the CHIME string format. The following example returns an array of molfile strings generated from both SMILES and CHIME string input.

Example 5 Converting structure types

```
package com.discoverygate.tutorial;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.discoverygate.webservice.DGWS;
import com.discoverygate.webservice.DiscoveryGate;
import com.discoverygate.webservice.DiscoveryGateLocator;
import com.discoverygate.webservice.types.DataPage;
import com.discoverygate.webservice.types.Datasource;
import com.discoverygate.webservice.types.MoleculeRetrievalFlags;
import com.discoverygate.webservice.types.MoleculeSearch;
import com.discoverygate.webservice.types.ServiceRequest;

public class Example5 {

  private static final String LICENSE_KEY = "<Your license key goes here>";

  /*
   * This method will create the minimal ServiceRequest object.
   */
  private static ServiceRequest createSr() {
    ServiceRequest sr = new ServiceRequest();
    sr.setLicenseKey(LICENSE_KEY);
    DataPage dp = new DataPage();
    dp.setOffset(0);
    dp.setCount(10);
    sr.setPage(dp);
    sr.setStatefulQueryKey("");
    Datasource[] sourceFilter = new Datasource[]{Datasource.CINDEX};
    sr.setSourceFilter(sourceFilter);
    return sr;
  }

  /**
   * @param args
   */
  public static void main(String[] args) {
    DiscoveryGate dg = new DiscoveryGateLocator();
    try {
      DGWS service = dg.getDGWS();

      final String  benzeneSMILES = "c1ccccc1";

      final String aspirinChime =
"CYAAFQwAYdwQyG^ZNk9YNlUstICQzJ8VVSYygQOg7qx2Ggw39BNeIHroIDNb2Of5FQswqsF9wiB3R5XuCTKOri4Ooh2
MyP4iN$GbwRbdUYewRddVU$qDIA25ER2t$WQKbdbSEhJaEojkRz69ibv3jp94XZLKRpIUNf1kkqlS$NJkUXqCR56oOUB
0ReWsRxSpqPItQ1MewK$uMFpSmeduAlPkjrT$$ixSKMCkjqDuHgAnG537hp3JNM8xilV9oPaW31e5U$UTnflRHWS3XXa
EMKv0$0I100ZG1EuST$uC6QRqvJv9cS7X7rv2Kvbcomf7VnlhUhv2qObT^mJU8Atp1W17jsHzV^JoTgIBkWSQPrypcnG
BX17$PfPgC4fzE^MErA";

      String[] structures = new String[] {benzeneSMILES, aspirinChime};
      String[] molfiles = service.convertStructureToMolfile(createSr(), structures);
      System.out.println("Structure conversion:");
      for(int i = 0; i < molfiles.length; i++) {
        System.out.println("'" + structures[i] +"' converts to :");
        System.out.println(molfiles[i]);
      }
    } catch (RemoteException e) {
      e.printStackTrace();
```



```
    } catch (ServiceException e) {
      e.printStackTrace();
    }
  }
}
```

Step 6: Retrieving and Caching Vocabularies

To ensure that your application has the latest version of data source vocabularies (see Vocabulary in the glossary), use methods such as getClassificationVocabulary, getProcurementVocabulary, getReactionVocabulary, and getToxicityVocabulary.

Your application can send the DGWS your vocabulary object. DGWS only returns a vocabulary if your current vocabulary is out of date.

Note: DGWS does not send the entire vocabulary, only what is needed to be up-to-date.

Example 6 - Vocabularies

The following example is not realistic since insofar as requests are made within the same process and it is unlikely that the version of the vocabulary changes. However, it does demonstrate a possible workflow:

```
package com.discoverygate.tutorial;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.discoverygate.webservice.DGWS;
import com.discoverygate.webservice.DiscoveryGate;
import com.discoverygate.webservice.DiscoveryGateLocator;
import com.discoverygate.webservice.types.DataPage;
import com.discoverygate.webservice.types.Datasource;
import com.discoverygate.webservice.types.MoleculeRetrievalFlags;
import com.discoverygate.webservice.types.MoleculeSearch;
import com.discoverygate.webservice.types.ReactionVocabulary;
import com.discoverygate.webservice.types.RxnVocabItem;
import com.discoverygate.webservice.types.ServiceRequest;
import com.discoverygate.webservice.types.VocabularyVersion;

public class Example6 {

  private static final String LICENSE_KEY = "<Your license key goes here>";

  /*
   * This method will create the minimal ServiceRequest object.
   */
  private static ServiceRequest createSr() {
    ServiceRequest sr = new ServiceRequest();
    sr.setLicenseKey(LICENSE_KEY);
    DataPage dp = new DataPage();
    dp.setOffset(0);
    dp.setCount(10);
    sr.setPage(dp);
    sr.setStatefulQueryKey("");
    Datasource[] sourceFilter = new Datasource[]{Datasource.CINDEX};
    sr.setSourceFilter(sourceFilter);
    return sr;
  }
```

```
  /**
   * @param args
   */
  public static void main(String[] args) {
    DiscoveryGate dg = new DiscoveryGateLocator();
    try {
      DGWS service = dg.getDGWS();

      // get the initial vocabulary
      ServiceRequest sr = createSr();
      ReactionVocabulary vocab1 = service.getReactionVocabulary(sr, new VocabularyVersion());
      System.out.println("Reaction Vocabulary:");
      long version1 = vocab1.getVersion();
      System.out.println("Version = " + version1);
      RxnVocabItem[] vocabItems = vocab1.getVocabulary();
      if(vocabItems != null) {
        for(RxnVocabItem item: vocabItems) {
          System.out.println("field = " + item.getField() + " - item = " + item.getItem());
        }
      }
      // now try it again
      ReactionVocabulary vocab2 = service.getReactionVocabulary(sr, vocab1);
      // this should be null now, since we don't anticipate any changes since the first call
      if(vocab2 == null) {
        System.out.println("No changes between last call and now!");
      } else {
        RxnVocabItem[] newVocabItems = vocab2.getVocabulary();
        if(newVocabItems == null) {
          System.out.println("No changes between last call and now!");
        } else {
          // this should not happen
          System.out.println("Updated version = " + vocab2.getVersion());
        }
      }
    } catch (RemoteException e) {
      e.printStackTrace();
    } catch (ServiceException e) {
      e.printStackTrace();
    }
  }
}
```

This example does the following:

1. Create a new empty vocabulary version and obtain the latest Reaction vocabulary from the service storing it in the local variable vocab1.

2. Then, check with the service to get a new vocabulary, if available, passing in the vocabulary version from the previously obtained vocabulary : `ReactionVocabulary vocab2 = service.getReactionVocabulary(sr, vocab1);`

   The vocabulary returned inside `vocab2` is either null (no change) and the version did not change, or it contains an array of the changed vocabulary items.

Advanced Techniques

Efficient Information Retrieval

Overview

DGWS implements a way to limit the amount of data being retrieved and transmitted over the wire through Retrieval Flags (Molecule Retrieval Flags and Reaction Retrieval Flags). Molecule and reactions flags control the data that the DiscoveryGate Web Service (DGWS) returns in Molecule and Reaction records. To improve performance and bandwidth usage by downloading only relevant data, use different retrieval flags.

Retrieval flags are being passed to each DGWS request as part of the ServiceRequest object as can be seen in Example4. There are two different retrieval flag objects, MoleculeRetrievalFlags and ReactionRetrievalFlags, and the Retrieval Flags should be added to the ServiceRequest as an array of those objects.

The different Retrieval Flags actually affect the details retrieved for each service call which is discussed in more details in the DGWS Developer's Guide chapters about Molecule Retrieval Flags and Reaction Retrieval Flags.

Stateful versus Stateless Searches

Generally speaking, as with most web services, DGWS is a stateless service. However, when using the special Retrieval Flag ID_ONLY (for MoleculeRetrievalFlag as well as for ReactionRetrievalFlag) method calls returning MoleculeResults or ReactionResults can be stateful.

What exactly does "stateful" mean in the context of DGWS?

As pointed out above, DGWS has been designed and implemented to return large result-sets in "chunks" rather than in one large block. For example, a substructure search (see Example3) may result in hundreds if not thousands of hits. Returning all those results together with all requested details by using Retrieval Flags may result in hundreds of kilo Bytes if not mega bytes being transferred over the wire back to your client. Not only will it consume a huge amount of bandwidth, it also may result in timeout situations, depending on your specific settings. So, instead DGWS will return the hits in "pages" of molecules or reactions and it is the client's responsibility to iterate over all pages to retrieve all the results page by page.

The other important question to discuss will be: what is the fundamental difference between stateful and stateless searches?

This is best answered using an example:

When running a stateful search, the only valid Retrieval Flag is ID_ONLY which will return only molecule IDs or reaction IDs but no details at all.

Let us assume we want to do a substructure search using the structure of 4-(1H-PYRAZOL-1-YL)BENZENESULFONAMIDE. This search should result in approximately 900 molecules (the exact number depends on the version of the underlying content databases). At a page size of 20 we will have to do approximately 45 calls using different offsets to retrieve all the results. If we use a stateless search, every call to getMoleculesByStructure to retrieve a page will actually end up doing the same search 45 times which is not very efficient.

On the other hand, when doing a stateful search, only the first call to getMoleculesByStructure will run the actual search retrieving all 900+ ids but only returning the first page (20). Each subsequent call to getMoleculesByStructure will simply retrieve the requested page without running the search again. Although the stateful search will only return IDs, it is more efficient to use it when the search is expected to run rather long and a large number of results is expected.

However, after running a stateful search, what can you do with the resulting IDs? The answer is actually pretty straight forward. You can use the IDs and pass them to a stateless call of getMoleculesById page by page and fortunately, the getMoleculesById is extremely efficient so using separate calls is actually better and faster than doing it all in one stateless getMoleculesByStructure call.

Example 7 - stateful and stateless searches

```
package com.discoverygate.tutorial;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.discoverygate.webservice.DGWS;
import com.discoverygate.webservice.DiscoveryGate;
import com.discoverygate.webservice.DiscoveryGateLocator;
import com.discoverygate.webservice.types.DataPage;
import com.discoverygate.webservice.types.Datasource;
import com.discoverygate.webservice.types.Molecule;
import com.discoverygate.webservice.types.MoleculeId;
import com.discoverygate.webservice.types.MoleculeResults;
import com.discoverygate.webservice.types.MoleculeRetrievalFlags;
import com.discoverygate.webservice.types.MoleculeSearch;
import com.discoverygate.webservice.types.MoleculeSearchType;
import com.discoverygate.webservice.types.Parenthesis;
import com.discoverygate.webservice.types.SearchOperator;
import com.discoverygate.webservice.types.ServiceRequest;

public class Example7 {

  private static final String LICENSE_KEY = "<Your license key goes here>";
  // The following represents 4-(1H-PYRAZOL-1-YL)BENZENESULFONAMIDE
  private static final String MOL_CHIME_STRING =
"CYAAFQwAUfwQBtB2BKbFUafalUsSO5MJJZaYDxVQKBN5FZaocjV5gksO8OASiQ$Gw$$miE4Ko3BSBEqn5BBkHeVE7xi
LdSYydecwxYxgjLfCy7F4QUyjDgs^cBQBOLXvIqM876HaZpRdUy0y9kpxAjbR^NSzawesl7TKTMJaRPsOIIPhnh5^qOP
F4F6Tyy8UkJwiF^MFPP6GdXxqamQQKTKp2n6vGR0VaXaunEiS1vbpo3mHqszCUVbPekC6OJczcz$sCbXau^AGyRI9ivL
pGxOe1HdVg^i7S5IpIKBGcHyQsqedSikoDWkCKPx9Ka5yFjp4KHjT1qoLxdYgqqcqqVrqM$jdsX9MNWVYgGcNr5Vqvyl
93Ddj2kLflG1A1RkmRZSbvuVXg3kabdh^P0cvEcZeMLDHAPT1rV602BwfnTOP";

  /*
   * This method will create the minimal ServiceRequest object.
   */
  private static ServiceRequest createStatefulSr() {
    ServiceRequest sr = new ServiceRequest();
    sr.setLicenseKey(LICENSE_KEY);
    DataPage dp = new DataPage();
    dp.setOffset(0);
    dp.setCount(20);
    sr.setPage(dp);
    // supply some arbitrary stateful key
    String statefulKey = String.valueOf(System.currentTimeMillis());
    sr.setStatefulQueryKey(statefulKey);
    MoleculeRetrievalFlags[] mrf = new
MoleculeRetrievalFlags[]{MoleculeRetrievalFlags.ID_ONLY};
    sr.setMoleculeFlags(mrf);
    Datasource[] sourceFilter = new Datasource[]{Datasource.CINDEX};
    sr.setSourceFilter(sourceFilter);
    return sr;
```



```
  }

  private static ServiceRequest createStatelessSr() {
    ServiceRequest sr = new ServiceRequest();
    sr.setLicenseKey(LICENSE_KEY);
    DataPage dp = new DataPage();
    dp.setOffset(0);
    dp.setCount(40);
    sr.setPage(dp);
    sr.setStatefulQueryKey("");
    MoleculeRetrievalFlags[] mrf = new MoleculeRetrievalFlags[]{};
    sr.setMoleculeFlags(mrf);
    Datasource[] sourceFilter = new Datasource[]{Datasource.CINDEX};
    sr.setSourceFilter(sourceFilter);
    return sr;
  }
  /**
   * @param args
   */
  public static void main(String[] args) {
    DiscoveryGate dg = new DiscoveryGateLocator();
    try {
      DGWS service = dg.getDGWS();

      MoleculeSearch[] molSearches = new MoleculeSearch[1];
      MoleculeSearch moleculeSearch = new MoleculeSearch();
      moleculeSearch.setStructure(MOL_CHIME_STRING);
      moleculeSearch.setSearchType(MoleculeSearchType.SUBSTRUCTURE);
      moleculeSearch.setBooleanOperator(SearchOperator.NONE);
      moleculeSearch.setParenthesis(Parenthesis.NONE);
      moleculeSearch.setCustomFlags("");
      molSearches[0] = moleculeSearch;
      ServiceRequest statefulSr = createStatefulSr();
      int pageSize = statefulSr.getPage().getCount();
      ServiceRequest statelessSr = createStatelessSr();
      int page = 1;
      do {
        MoleculeResults mrStateful = service.getMoleculesByStructure(statefulSr,
molSearches);
        int count = mrStateful.getContainedCount();
        int overallCount = mrStateful.getQueryCount();
        System.out.println("We got " + count + " result(s).");
        System.out.println("Query count is " + overallCount);
        Molecule[] molecules = mrStateful.getRecords();
        MoleculeId[] molIds = new MoleculeId[count];
        for(int i = 0; molecules != null && i < count; i++) {
          long id = molecules[i].getId();
          // now we gather all the retrieved IDs into an array
          MoleculeId molId = new MoleculeId(id);
          molIds[i] = molId;
        }
        // now make another stateless call to get the molecule details passing in our molIds
        MoleculeResults mrStateless = service.getMoleculesById(statelessSr, molIds);
        System.out.println("Results from page " + page + ":");
        for(int i = 0; i < molIds.length ; i++) {
          System.out.println("Molecule #" + i + ": mol formula = '" +
mrStateless.getRecords(i).getMolformula() + "'");
        }
        if(overallCount > 0) {
          // we got all pages, lets exit this loop
```

```
        System.out.println("Retrieved all " + overallCount + " results - exiting");
        break;
       }
       int currentOffset = statefulSr.getPage().getOffset();
       statefulSr.getPage().setOffset(currentOffset + pageSize);
       page++;
     } while(true);
     // when done, we should ALWAYS call closeStatefulQuery using our stateful
ServiceRequest
     service.closeStatefulQuery(statefulSr);
   } catch (RemoteException e) {
     e.printStackTrace();
   } catch (ServiceException e) {
     e.printStackTrace();
   }
  }

}
```

There are a few important facts to note about stateful searches:

1. The result of a statefule search will have to be cached on the server until all IDs have been retrieved by the client. Since this will consume shared resources, a stateful query will automatically be closed after 60 seconds of inactivity!

2. The search criteria provided in the ServiceRequest object cannot be changed after the first service call. Doing this may have unpredictable results!

3. After the first stateful search, the offset in the given data page can only be increased. If the offset are being decreased, i.e. retrieving results from previous pages, the search will have to be run again as each ID that has been retrieved by the client will be cleared from the underlying cache and is not accessible anymore without re-doing the search.

