The CSD Pipeline Pilot Component Collection

2020.0 CSD System Release

Conditions of Use

The Cambridge Structural Database System (CSD System) comprising all or some of the following:

ConQuest, Quest, PreQuest, deCIFer, Mercury, (Mercury CSD and CSD-Materials [formerly known as the Solid Form or Materials module of Mercury], Mercury DASH), Mogul, IsoStar, DASH, SuperStar, web accessible CSD tools and services, WebCSD, CSD Java sketcher, CSD data file, CSD-UNITY, CSD-MDL, CSD-SDfile, CSD data updates, sub files derived from the foregoing data files, documentation and command procedures, test versions of any existing or new program, code, tool, data files, sub-files, documentation or command procedures which may be available from time to time (each individually a Component) is a database and copyright work belonging to the Cambridge Crystallographic Data Centre (CCDC) and its licensors and all rights are protected. Use of the CSD System is permitted solely in accordance with a valid Licence of Access Agreement or Products Licence and Support Agreement and all Components included are proprietary. When a Component is supplied independently of the CSD System its use is subject to the conditions of the separate licence. All persons accessing the CSD System or its Components should make themselves aware of the conditions contained in the Licence of Access Agreement or Products Licence and Support Agreement or the relevant licence.

In particular:

- The CSD System and its Components are licensed subject to a time limit for use by a specified organisation at a specified location.
- The CSD System and its Components are to be treated as confidential and may NOT be disclosed or redistributed in any form, in whole or in part, to any third party.
- Software or data derived from or developed using the CSD System may not be distributed without prior written approval of the CCDC. Such prior approval is also needed for joint projects between academic and for-profit organisations involving use of the CSD System.
- The CSD System and its Components may be used for scientific research, including the design of novel compounds. Results may be published in the scientific literature, but each such publication must include an appropriate citation as indicated in the Schedule to the Licence of Access Agreement or Products Licence and Support Agreement and on the CCDC website.
- No representations, warranties, or liabilities are expressed or implied in the supply of the CSD System or its Components by CCDC, its servants or agents, except where such exclusion or limitation is prohibited, void or unenforceable under governing law.

Licences may be obtained from:

Cambridge Crystallographic Data Centre
12 Union Road
Cambridge CB2 1EZ, United Kingdom

Web: http://www.ccdc.cam.ac.uk
Telephone: +44-1223-336408
Email: admin@ccdc.cam.ac.uk

(UNITY is a product of Certara and MDL is a registered trademark of BIOVIA)

Contents

1 Introduction

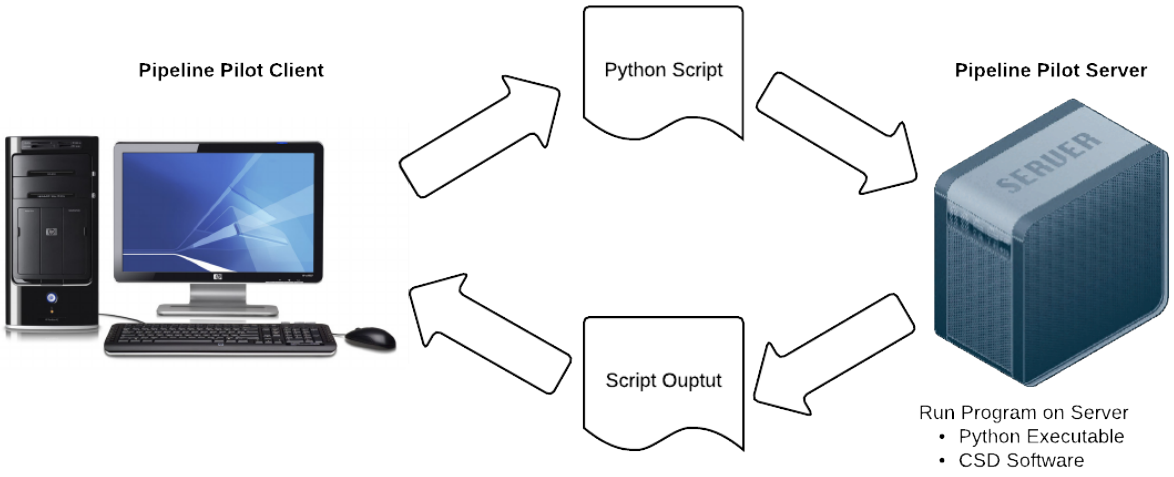
This component collection for Pipeline Pilot allows protocols to integrate functionality from the Cambridge Structural Database System (CSD) Python API. This includes capabilities for CSD searching, model validation, conformer generator and virtual screening. There are also components to help in the integration of further Python scripts which may use the CSD Python API.

Please do not hesitate to contact support@ccdc.cam.ac.uk for more information and help.

1.1 Integration Scheme

The integration works as follows:

- The components found in the CSD Pipeline Pilot (PP) Component Collection create the necessary input files from the incoming data, and from the parameters on those components.
- The components find the appropriate Python script in the component collection; various python scripts are supplied.
- The *Run Program on Server* component runs the Python executable, supplying the script and arguments for that script, which produce the output.
- The components then read the file(s) generated by the script, performing appropriate merges, joins, etc. to produce a stream of data.



1.2 Requirements

The Pipeline Pilot server host must have installed:

- Pipeline Pilot server (version 19.1 or later).
- The 2020 release of the CSD, including required licenses.
    - Currently supported operating systems are Windows Server 2019 and CentOS 7, although CentOS 8 and RHEL 7 and 8 are also expected to work.
    - The minimum specifications for the CSD Python API apply.

On Linux, the user which the Pipeline Pilot server runs as must have a home directory for the CSD licensing to function.

To use the Mercury Viewer or Hermes Viewer components, Mercury and Hermes must be installed on the client. A free version of Mercury can be downloaded here.

1.3 Installation of the CSD Pipeline Pilot Component Collection package

Installation on Windows

All the actions described below should be carried out on the PP server host.

1. Ensure that all the requirements specified above are met.

2. If you have a previous version of the package installed, it must be uninstalled first (see below).

3. Open a PowerShell prompt as Administrator and navigate to the 'apps' folder under the PP server installation. If the PP server has been installed into the default location, this would be 'C:\Program Files\BIOVIA\PPS'.

4. In the 'apps' folder, inflate the package zip archive. A folder 'ccdc' should then be present.

5. Activate the CSD Python API environment. Assuming the CSD has been installed into the default location, use the commands:

```
. "C:\Program Files\CCDC\Python_API_2020\miniconda\shell\condabin\conda-hook.ps1"

conda activate "C:\Program Files\CCDC\Python_API_2020\miniconda"
```

6. Use the 'pkgutil' tool to install the package; this registers protocols and components and makes them visible to the Pipeline Pilot client (*N.B.* a forward slash is required in the package name):

```
..\bin\pkgutil.exe -i ccdc/pythonapi
```

Installation on Linux

All the actions described below should be carried out on the PP server host.

1. Ensure that all the requirements specified above are met.

2. If you have a previous version of the package installed, it must be uninstalled first (see below).

3. Open a terminal and navigate to the 'apps' directory under the PP server installation. A typical location for the PP server on Linux would be '/opt/BIOVIA/PPS'.

4. In the ‘apps’ directory, inflate the package zip archive. A directory ‘ccdc’ should then be present.

5. Activate the CSD Python API environment. If the installation prefix for the CSD System was ‘/opt/CCDC’, the appropriate command would be:

```
. /opt/CCDC/Python_API_2020/miniconda/bin/activate
```

6. Set the CSDHOME environment variable. If the installation prefix for the CSD System is as above, the appropriate command would be:

```
export CSDHOME=/opt/CCDC/CSD_2020
```

7. Set up the environment for the ‘pkgutil’ tool:

```
. ../linux_bin/ppvars.sh
```

8. Use the ‘pkgutil’ tool to install the package; this registers protocols and components and makes them visible to the Pipeline Pilot client:

```
../linux_bin/pkgutil -i ccdc/pythonapi
```

IMPORTANT: You may see an error message like the following:

```
ccdc/pythonapi was not installed:
   Unable to open file
<pps_dir>/apps/ccdc/pythonapi/docs/pipeline_pilot_component_collection.htm
l-tmp for reading: No such file or directory
```

This is a problem with Pipeline Pilot, and while HTML documentation will not have been generated, the package itself should actually have been installed.

Configuration

Once the package has been installed, the PYTHON_HOME Global Property needs to be set to the path of the CSD Python API directory containing the python executable. This is done using the web-based Administration Portal, accessible from the Server Home Page (see the Help menu of the PP client). Note that the default username is ‘scitegicadmin’ and the password ‘scitegic’.

1. Under ‘Admin Pages’, open the ‘Setup’ folder and click on ‘Global Properties’.

2. From the Package drop-down, select ‘CCDC/CSD Pipeline Pilot Collection’

3. Select the ‘PYTHON_HOME’ property by clicking on the name and set the value to the appropriate path. If the CSD installation is as above, this would be:

Windows: `"C:\Program Files\CCDC\Python_API_2020\miniconda"`

Linux: `/opt/CCDC/Python_API_2020/miniconda/bin`

All the other values can be left blank.

Uninstallation on Windows

1. Open a PowerShell prompt as Administrator and navigate to the PPS 'apps' folder.

2. Use the 'pkgutil' tool to uninstall the package:

```
..\bin\pkgutil.exe -u ccdc/pythonapi
```

3. Delete the 'ccdc' folder.

Uninstallation on Linux

1. Open a terminal and navigate to the 'apps' directory under the PP server installation.

2. Set up the environment for the 'pkgutil' tool:

```
. ../linux_bin/ppvars.sh
```

3. Use the 'pkgutil' tool to uninstall the package:

```
../linux_bin/pkgutil -u ccdc/pythonapi
```

4. Remove the 'ccdc' folder.

1.4 About the CSD PP Component Collection

This collection was developed in partnership with Finia Consulting.

Finia Consulting was founded in 2013 and is a small software consultancy, specialised in using the Pipeline Pilot platform to develop complex protocols and components including custom components developed using Java and C#, and can also provide custom and specialized training for the Pipeline Pilot platform. Finia Consulting have customers in the life sciences, chemicals and academic sectors.

Finia Consulting may be contacted at:

Web: www.finiaconsulting.com

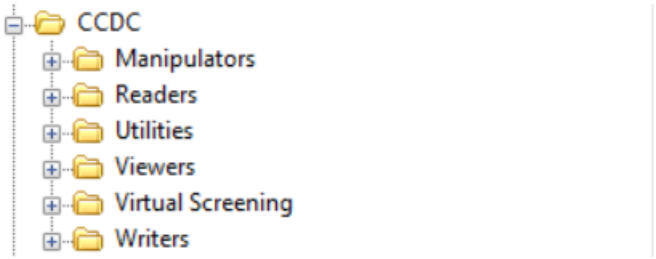
Email: pcochrane@finiaconsulting.com

Phone: +44 118 981 5993

2 CSD Python API Components

The CSD Component Collection consists of a series of components designed for regular usage. These are underpinned by utilities components which can be used to perform lower-level operations. For most day to day usage, it is expected that the higher-level components will be sufficient. A series of protocols are provided with CSD PP Component Collection that show how to combine the different components in a workflow and allow to perform several complex operations such as merged, compared, and processed, according to the logic of the protocol.

The components provided within the CSD PP Component Collection are included in the `CCDC` folder in the PP Components. The CCDC components are organised in broader categories related to the type of operation performed by the included components such as: **Manipulators, Readers, Utilities, Viewers, Virtual screening and Writer.** Note that the **Virtual Screening** and the **Writers** components are considered as high-level components and are available for CSD-Discovery and CSD Discovery or CSD-Material users only.



2.1 Manipulators

The manipulator folder includes components designed to make specific changes to select data record properties.

Convert Unicode Characters

The *Convert Unicode Characters* component converts the unicode strings found in one or more properties into their appropriate character.

For example, consider the value:

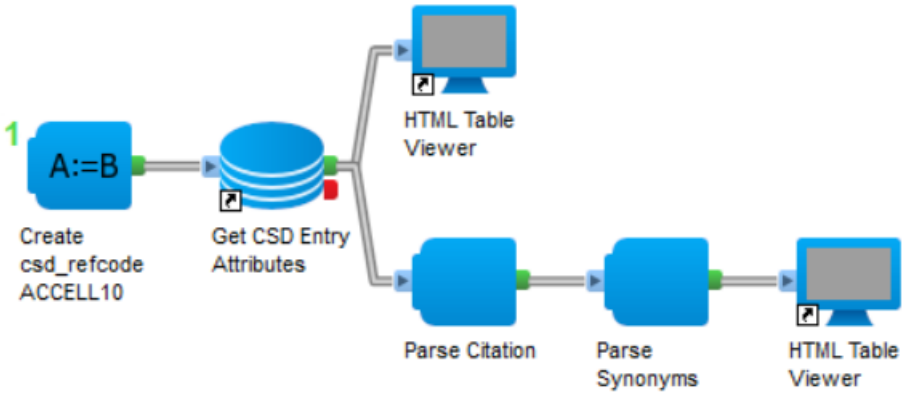
```
6'-O-Trityl-\u03b1-cellobiose hepta-acetate
```

This contains the unicode character `\u03b1`. To convert this to its character (using the `Chr()` function), this must first be converted from the HEX value to a decimal number. In this case, that's 945. This is the alpha character α. This makes this property.

```
6'-O-Trityl-α-cellobiose hepta-acetate
```

Parse Citation & Parse Synonyms

The publication or citation properties, if retrieved, are defined as an encoded Python string. It's more or less readable, but contains markup related to how Python converts objects and lists to string. The *Parse Citation* component translates those strings to separate properties for each value found in the citation. In the same way, the Parse *Synonym* component translates the synonym property, if retrieved from the encoded Python string, to more formal strings, and where multiple synonyms are found, converts the property into an array.



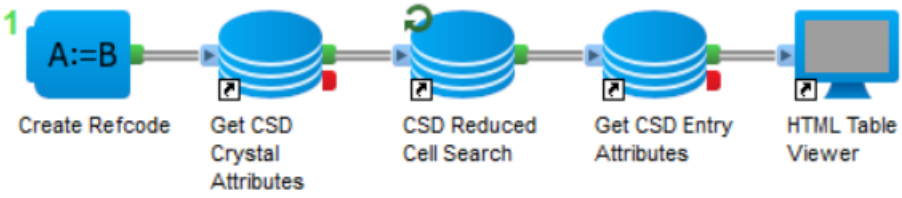
In the example above, we take citations and produce separate properties for each field in the citation, and we take the synonyms and convert them into arrays.

2.2 Readers

A reader is a component that generates a stream of data records that are then pushed through subsequent components in a pipeline. The data is based on input from a data source — usually a file or database. Data readers are frequently used as the initial component in a pipeline.

Get CSD Crystal Attributes

The *Get CSD Crystal Attributes* component allow to retrieve crystal attributes for a series of CSD refcodes. This correspond to gathering the attributes found on the `ccdc.crystal.Crystal` CSD Python API module. The `ccdc.crystal.Crystal` class contains attributes relating to the crystal structure for the entry, e.g. crystal structure details such as cell lengths, cell angles, and lattice centring information. These details can be used in a reduced cell search.



CSD Reduced Cell Search

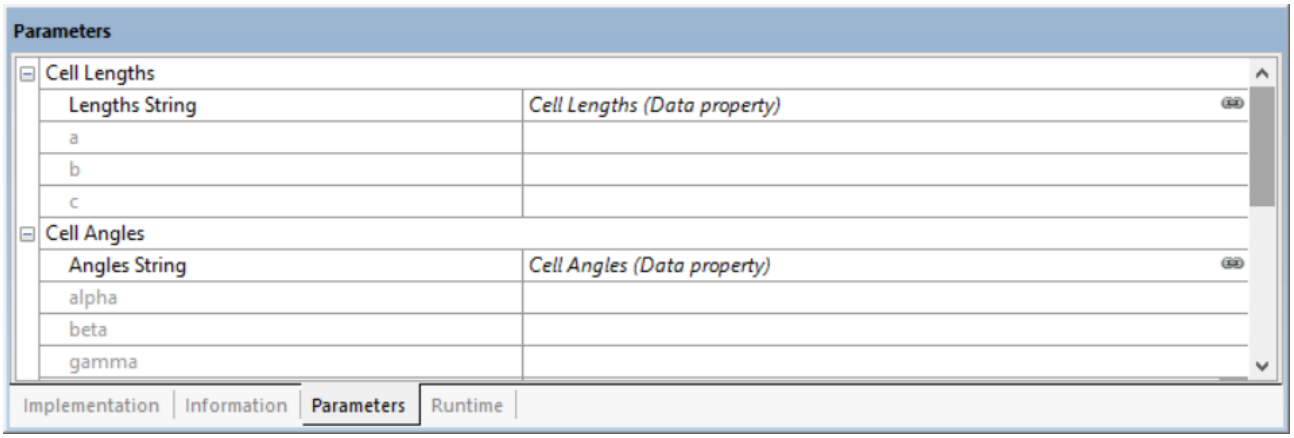
The *CSD Reduced Cell Search* component performs a reduced cell search of the CSD. Reduced cell searches can be carried out in two ways.

In the example above, the cell lengths and angles returned by the crystal attributes are used. These are strings of the form:

```
CellLengths (a=8.4708, b=10.0492, c=14.0363)
```

```
CellAngles (alpha=86.016, beta=79.914, gamma=71.818)
```

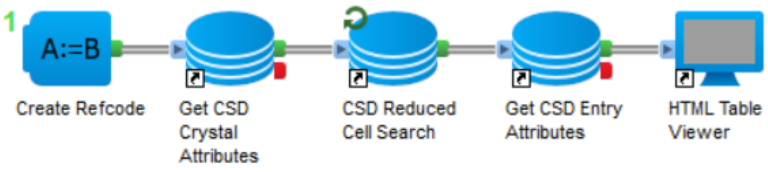
The alternative is to enter the `a`, `b`, `c`, and `alpha`, `beta`, `gamma` values separately in the appropriate parameters.



With the refcodes found by this search, the chemical name is retrieved and passed to the *Get CSD Entry Attributes* component.

Get CSD Entry Attributes

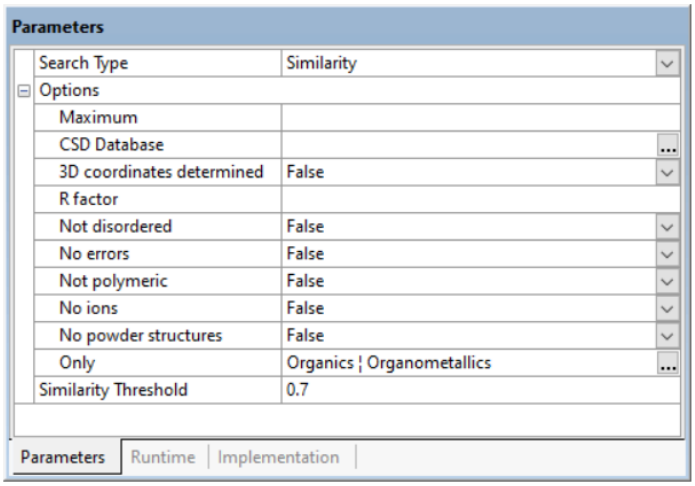
Get CSD Entry Attributes component retrieves entry attributes. In the example below it will provide the entry attributes for the reduced cell search results. This corresponds to gathering the attributes found on the `ccdc.entry.Entry` CSD Python API module. The `ccdc.entry.Entry` class contains attributes relating to the entry in the CSD, for example the citation information, chemical name and activity.



CSD Similarity Structure Search

CSD Similarity Structure Search component performs a similarity search on the CSD. Incoming molecules are treated as queries (mol2 and sdf format are supported). The records for which CSD compounds are found is output with the CSD refcode, along with the data from the incoming record for which this refcode was a hit. This can be useful when passing in multiple queries.

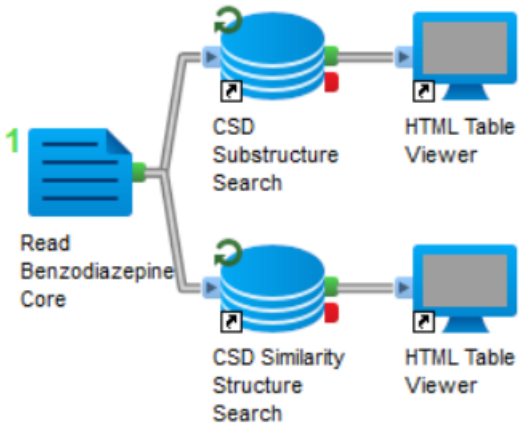
Similarity searches take into account the similarity threshold which all hit structures must exceed. The similarity threshold and the search filters can be edited in the **Parameters** section of the component.



To aid in the interpretation of the results, the component outputs have an additional property which is the similarity value for this compound to the query for which it was found.

CSD Substructure Search

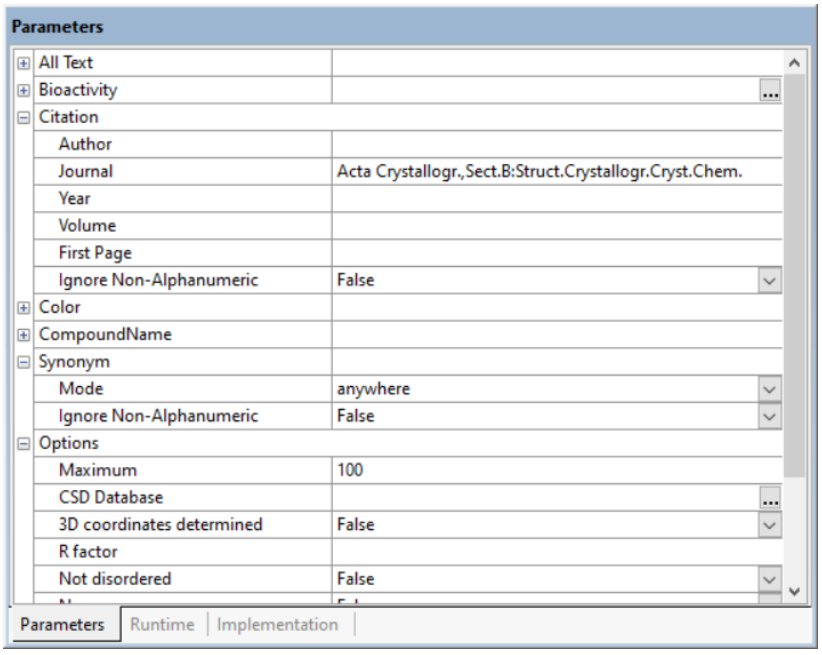
CSD Substructure Search component performs a substructure or exact match search on the CSD. Incoming molecules are treated as queries (mol2 and sdf format are supported). The records for which CSD compounds are found, is output with the CSD refcode, as well as the data for the incoming query structure for which this was a hit. This can be useful when passing in multiple queries.



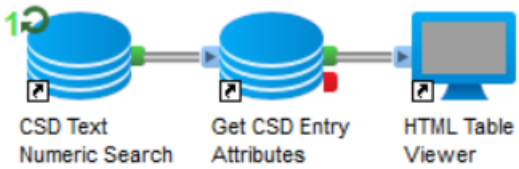
The example above shows how the *CSD Substructure Search* and *CSD Similarity Structure Search* can be combined to read a mol file and perform a substructure and a similarity search in CSD, retrieving the results in two distinct HTML tables.

CSD Text Numeric Search

CSD Text Numeric Search component performs a text numeric search against the CSD and produce a stream of CSD refcodes for the hits found by the search. The query is built up from the criteria entered in the component **Parameters**.



Each element is combined in an AND fashion. For example, specifying an author name and a journal will only find an entry with the specified author is in the specified journal.

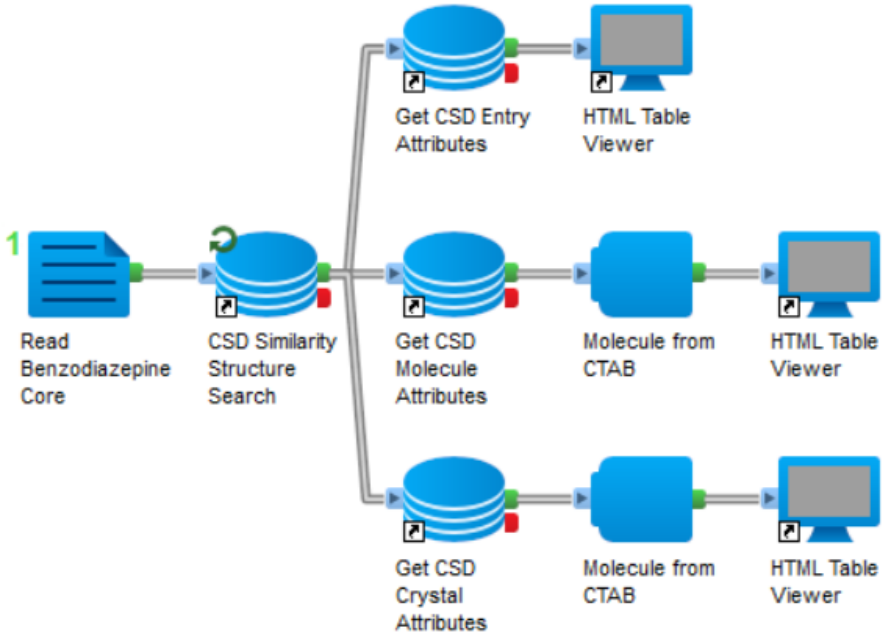


In the example above, the *CSD Text Numeric Search* component is used to search the CSD for all crystals reported in *Acta Crystallogr.,Sect.B:Struct.Crystallogr.Cryst.Chem*. journal limiting the search to 100 records by setting the **Maximum** parameter in the **Options** section of the **Parameters** section of the component.

Get CSD Molecule Attributes

Get CSD Molecule Attributes component retrieves molecule attributes for a set of CSD refcodes. This corresponds to gathering the attributes found on the `ccdc.molecule.Molecule` entry of the CSD Python API. The `ccdc.molecule.Molecule` class contains attributes relating to the chemistry of the molecule, for example the SMILES representation.

In some cases, a molecule may not have a canonical SMILES representation e.g. where the structure has unknown atoms or bonds, `AJABIX01` for example. In such cases, the SMILES property will be removed to avoid attempting to interpret None as a SMILES.



In the example above the *Get CSD Molecule Attributes* component is used in combination with the *Get CSD Entry Attributes* and *Get CSD Crystal Attributes* to retrieve for each entry (derived from the similarity search), not only the chemistry and the chemical attributes associated with the entry but also the publication details of a CSD entry and the associated DOI.

Get Molecule Structure

Get Molecule Structure component, gets the structure from the CSD as a molecular object which Pipeline Pilot can understand and use. This is just a wrapped version of the *Get CSD Molecule Attributes* component, with the **Attributes** requested set to *"CTAB"*. The CTAB is then converted into a molecular object. The refcode is assumed to be unique across all data sources available (e.g. same refcode will not occur in data source 2 if it came from data source 1). Therefore, the database selection is limited to one data source - to avoid looking across multiple data sources, only one of which will actually contain the refcode.

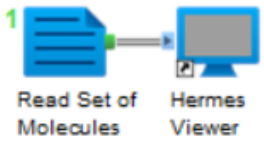
2.3 Viewers

A viewer is a component that displays information or results on your client. Viewers are frequently used as the final component in a pipeline however, they can be also used to view intermediate results.

In addition to the generic viewer provided by Pipeline Pilot, the CSD PP Component Collection includes four viewer components that are useful when using the CSD components.

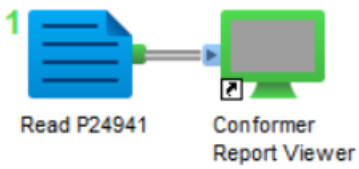
Hermes Viewer & Mercury Viewer

Hermes Viewer & Mercury Viewer components allow to view the incoming stream of data in Hermes and Mercury, respectively. To aid in identifying the structures loaded, the name of the MOL2 file which will be passed to Hermes or Mercury can be supplied using the **Dataset Name** option of the component's parameters or can be provided as a reader component as showed in the example below.

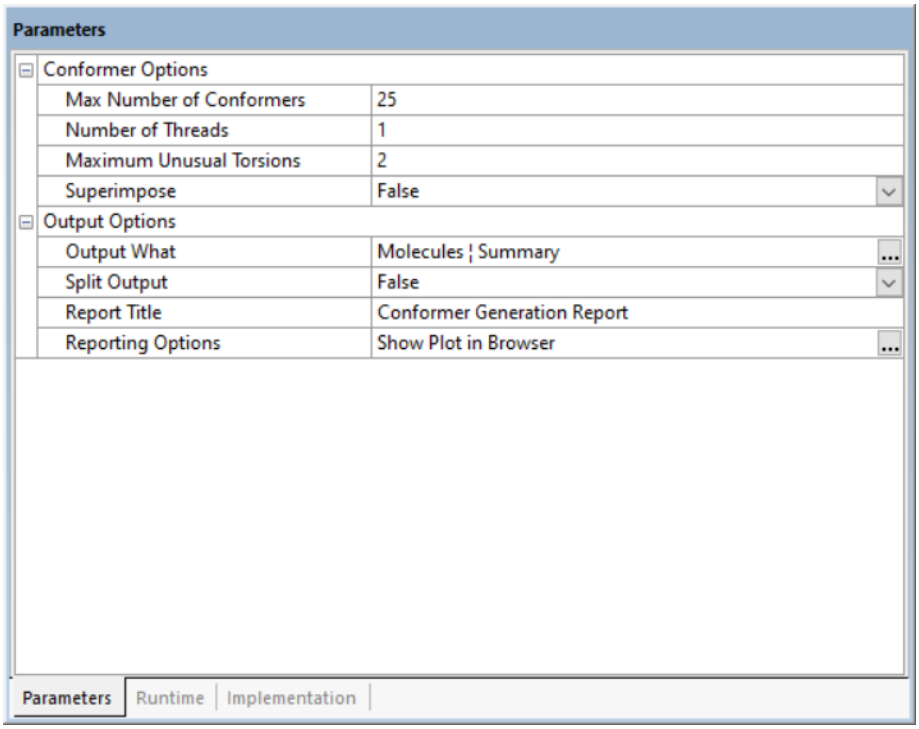


Conformer Report Viewer

Conformer Report Viewer component is available only for users with a CSD-Discovery and/or a CSD-Material licences. This component generates conformers for the incoming file of molecule(s) (SDF and MOL2 formats are supported) and produces a report summarizing the process. This means that the component will return a HTML summary of the settings used, a summary of the conformer generation results and links to the file(s) produced. By default, the summary file and the conformers outputs are generated.

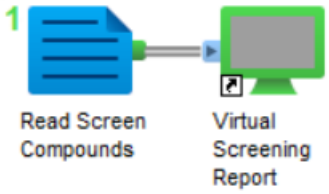


From the **Parameters** section of the *Conformer Report Viewer* component it is possible to change both the **Conformer Options** such as **Max Number of Conformers** and the **Output Options** of the component.

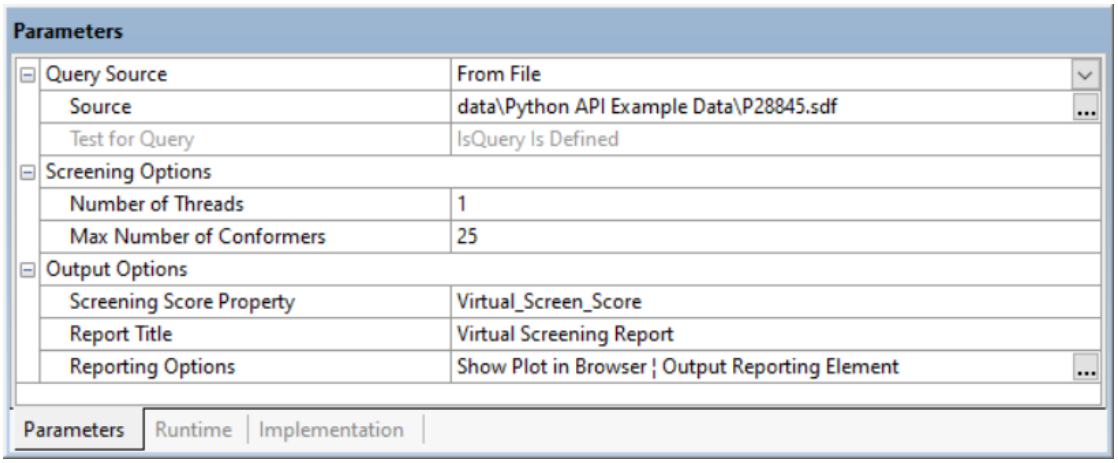


Virtual Screening Report Viewer

Virtual Screening Report Viewer component is available only for users with a CSD-Discovery licence. This component produces an HTLM report from the virtual screening process. This component receives a stream of incoming molecules and screens against a query set to generate a virtual screening score.



The query set may be supplied as file, specifying the **Source** in the **Parameters** section of the component. The query set file can be supplied either as a MOL2 or SD file.



The *Virtual Screening Report Viewer* component may also receive a stream of query records which are tagged to differentiate them from the screening set. This is achieved by setting the **Query Source** parameter to "*From Tag"* and declaring the PilotScript which differentiates the query records from the screening records.

The resulting HTML report contains the settings used for the virtual screening, plus links to the files used (screening and query), as well as the results file produced.

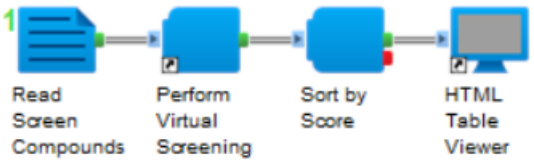
The *Virtual Screening Report Viewer* component can be used as part of a larger report by using the option to output reporting elements setting in the **Reporting Options** parameter of the component to include "*Output Reporting Element"*.

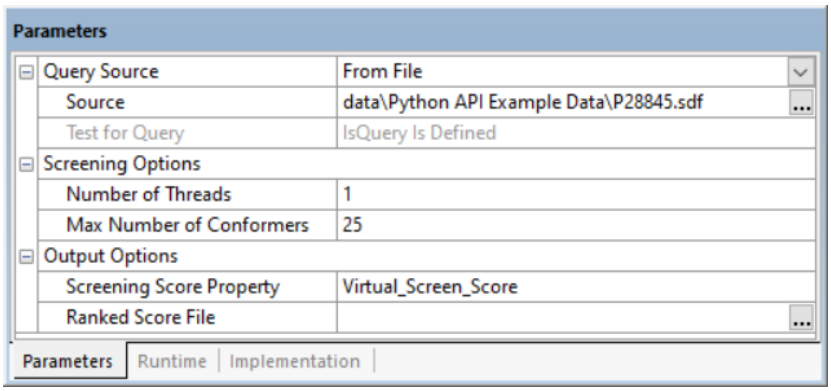
2.4 Virtual Screening

The **Virtual screening** components includes different components useful to perform ligand based virtual screening but also components that allow to assess the validity of the methodology. Note that *Perform Virtual Screening, Perform Virtual Screening Validation*, *Generate Enrichment Plot* and *Generate ROC Plot* are only available for users with a CSD-Discovery licence while the *Perform Conformer Generation* component is available for users with CSD-Discovery and/or CSD-Material licence.

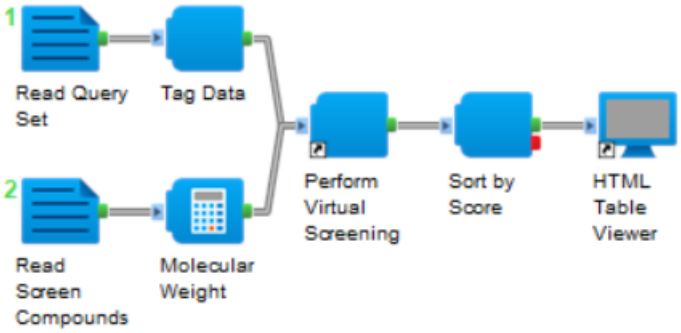
Perform Virtual Screening

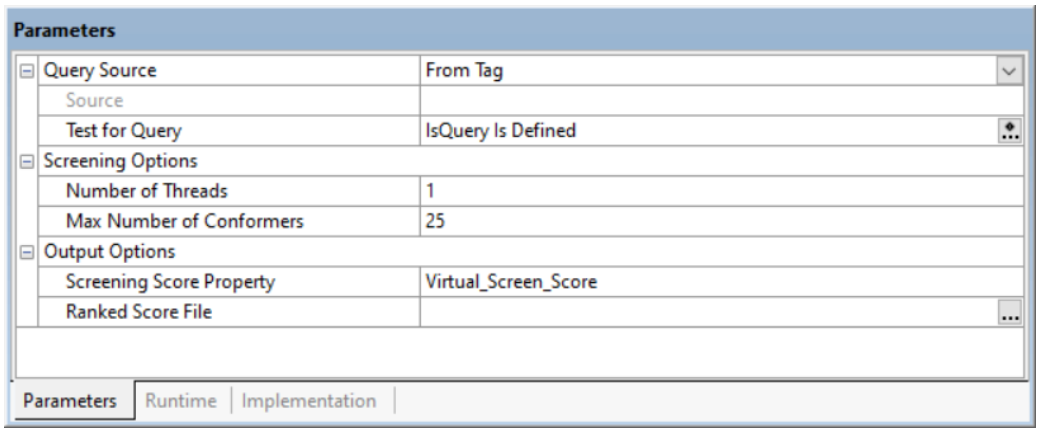
This component receives a stream of incoming records and applies a set of query molecules to that screening set to generate a virtual screening score. The query set may be supplied either as a MOL2 or SD file, specified using the **Source** parameter in the component, or, it may receive a stream of query records which are tagged to differentiate them from the screening set. This is achieved by setting the **Query Source** parameter to "*From Tag"* and declaring the PilotScript which differentiates the query records from the screening records.





The above example shows the component being used in a manner where the query file already exists and is specified in the **Source** parameter.





However, as the above example demonstrates, the stream of records can arrive at the component with records tagged appropriately. The component then internally divides the records according to the PilotScript found in **Test for Query** to produce a query set and screening set.

Whether acting on a pre-existing query source file or dynamically generating the query set by tag, the component has the same control over the underlying Python script. These are exposed in the **Screening Options** group parameters.

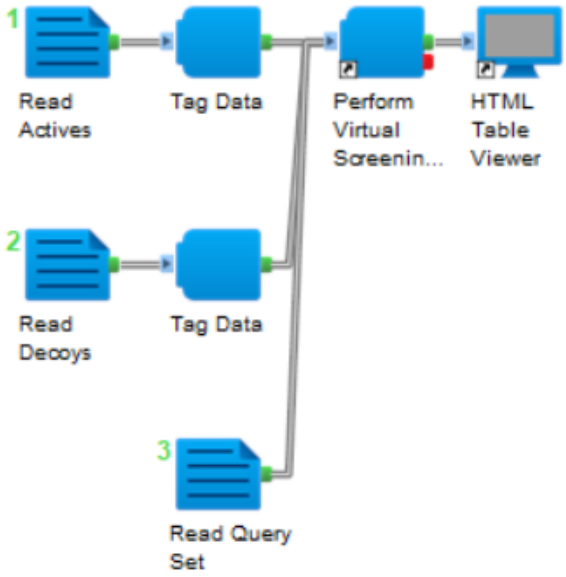
Finally, the data which is output by the component has a new virtual screening score which must be stored in a property. The component allows the user to set the name of this property using **Screening Score Property** parameter.

Perform Virtual Screening Validation

The *Perform Virtual Screening Validation* component takes a stream of query records and compares these to known actives and known decoys. This is used to assess the selectivity of the screen by producing a stream of the actives and decoys with scores. The scores are sorted from lowest to highest, where the lowest being the best and the occurrences of genuine actives *vs* decoys are tagged to help identify them.

As with the *Perform Virtual Screening* component, this component can work with actives and decoys in files specified in the appropriate source parameter, or it can identify the actives and/or decoys in-line.

The data produced has two new properties; one for the active tag (a value of 1 in this property implies the record related to the active set) and another for the virtual screening score. Therefore, the **Output Options** group has parameters to control the naming of these properties; **Active Tag** and **Screening Score Property** respectively.



Parameters

Input Options	
Active Source	From Tag
Source	
Test for Active	IsActive Is Defined
Decoy Source	From Tag
Source	
Test for Decoy	IsDecoy Is Defined
Screening Validation Options	
Number of Threads	1
Max Number of Conformers	25
Output Options	
Active Tag	Is_Active
Screening Score Property	Virtual_Screen_Score

Parameters | Runtime | Implementation

The above example shows the use of the *Perform Virtual Screening Validation* component where both the actives and decoys are identified by tag - both **Active Source** and **Decoy Source** are set to "*From Tag"*, with the **Test for Active** being the PilotScript "*IsActive Is Defined*", whilst the **Test for Decoy** parameter expressing the PilotScript "*IsDecoy Is Defined*". Anything which neither tag is declared is assumed to be a query record.

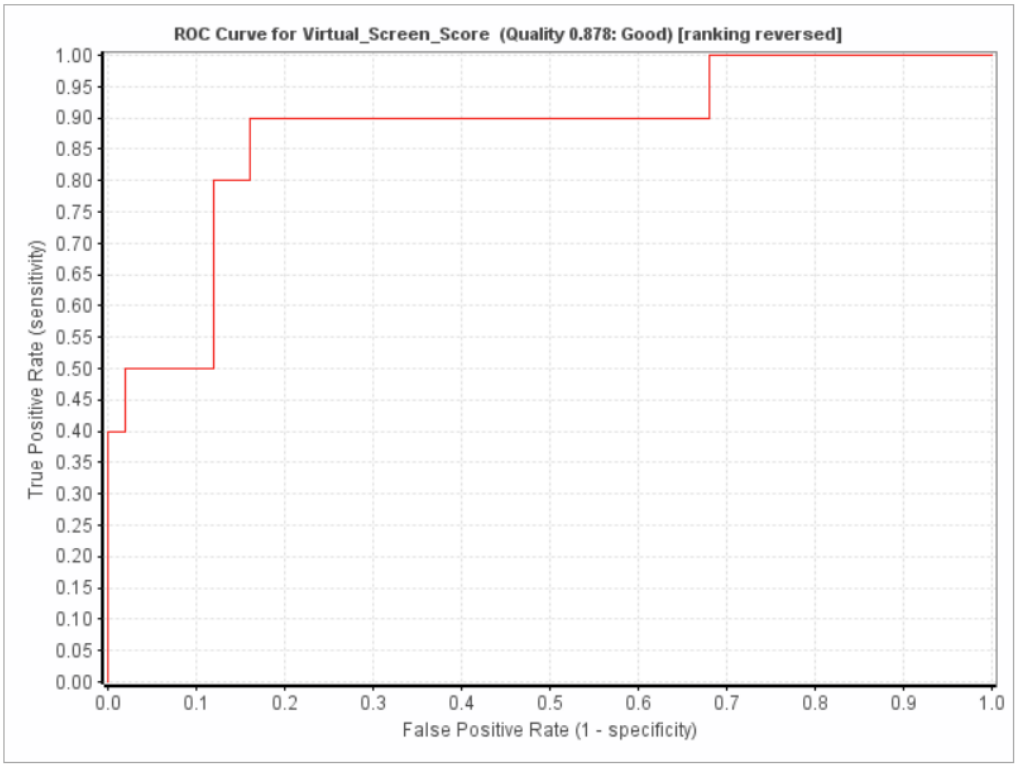
As with the previous component, the script arguments are expressed as options in the **Screening Validation Options** group parameters.

Finally, the **Output Options** include the names to be given to the **Active Tag** ("*Is_Active***"** in the example above) and to the **Screening Score Property**.

Generate Enrichment Plot & Generate ROC Plot

The *Perform Virtual Screening Validation* component produces "raw" data, data which is more normally presented to the use in a visual form. Therefore, the *Generate Enrichment Plot* component and *Generate ROC Plot* component exist to present that data in a chart. To that extent, both components are simple extensions of the *Perform Virtual Screening Validation* component, exposing almost identical parameters to the user.

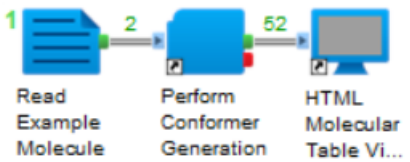
We can then use the *Generate ROC Plot* component to generate the ROC plot for the virtual screen study, that together with *Generate Enrichment Plot* components provides information about the selectivity of your model.



The main difference between the basic validation component and *Generate Enrichment Plot* and *Generate ROC Plot* components is that these components can either produce a chart in the browser or produce reporting component which can be incorporated as part of a larger report.

Perform Conformer Generation

The *Perform Conformer Generation* component is only available for user with CSD-Discovery or CSD-Material licence. The *Perform Conformer Generation* component takes a stream of molecular records and generates multiple conformations for each structure. The component outputs a stream of data where each record represents a conformer for an individual. The number of conformers is limited by the **Max Number of Conformers** in the **Conformer Options** parameter.



In the above example, two records are read in, and for each record, up to twenty-five conformers are generated (default value). Note that if the conformer generation script cannot find the maximum number of conformers, then it will output the number if finds. So, with a higher limit on the maximum number, it's possible to see different numbers of records per starting record. This conformer generation is also found under the virtual screening processes. As with the virtual screening components, this component will ensure that any data on the incoming data is re-attached to the output data.

2.5 Utilities Components

There are four internal utilities components that can be used to perform lower-level operations. These include:

- Run Python Script
- Run Virtual Screening
- Run Virtual Screening Validation
- Run Conformer Generation

The first component is capable of running any Python script, whilst the second runs the virtual screening script, and the third runs the virtual screening validation script. The final component generates conformers. These are designed to mirror the execution of these scripts on the command line.

Run Python Script

The *Run Python Script* is used internally, either directly or indirectly, in all components and protocols found in the CSD PP Component Collection package. The package contains a single global variable (e.g . @/ccdc/pythonapi/python_home) which declares the directory in which the python executable is found or installed on the server.

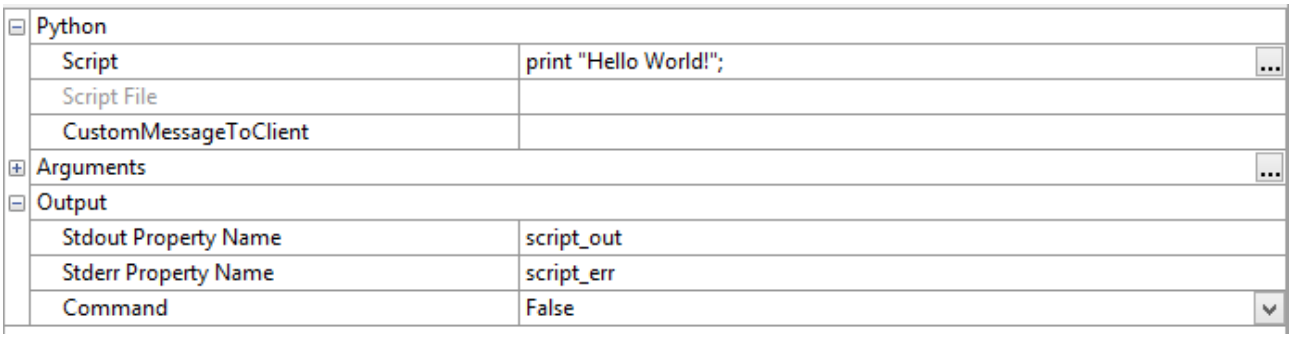
This component is intended for simple usage, and to test that the package is configured correctly. The *Run Python Script* component uses the @/ccdc/pythonapi/python_home global variable, combined with either a script file, or the content of a script which is written out as a file, together with the arguments to perform whatever task the script is designed to do.

The *Run Python Script* component produces potentially three properties, standard out for the script, standard error for the script and the command executed. The final option is useful for debugging as it can reveal when the script was invoked incorrectly.

The script to be executed can be specified by defining the entire script in the **Script** parameter declared in the **Python** group parameters. The script is entered here as any python script would be written. Internally, the component writes this script out to a file before executing it. Alternatively, the script can be specified as being in a file, located by the **Script File** parameter in the same group. Note that either one or the other of these two parameters must be supplied.

As it is often the case that a script may take a significant amount of time to run, the component exposes a **CustomMessageToClient** parameter to set the message to display whilst the script is running.

The **Arguments** group parameter is an array group which allows the creation of multiple arguments to be supplied to the python script. The arguments take on a pairing of **Switch** and **Value** - where switch is supplied first, followed by the value. Note that group arrays of this type natively support reordering if the order of arguments is important.

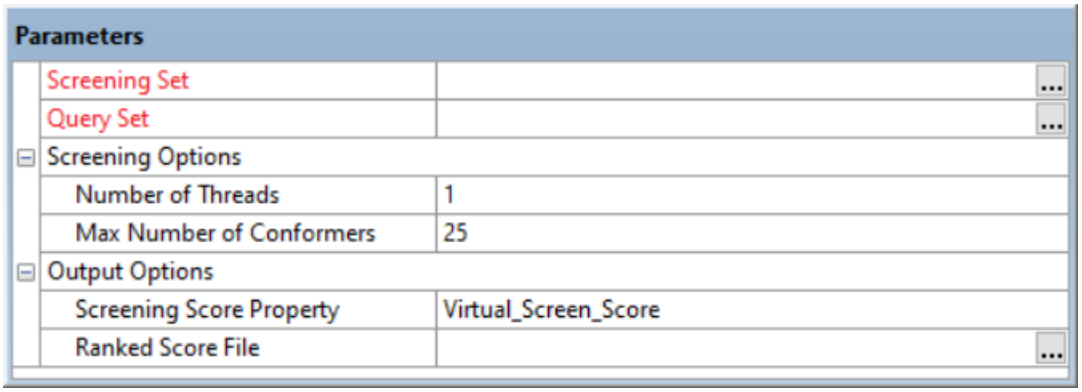


The above example shows a very simple Python script, supplied as a script in the **Script** parameter. This will produce the very simple ***"Hello World!"*** message in the property specified in the **Stdout Property Name** *"script_out*".

Run Virtual Screening

This component is a low-level wrapper over the Python script which performs the screening operation. The *Run Virtual Screening* component performs the screening operation found in the *Perform Virtual Screening* component.

Whereas *Perform Virtual Screening* component handles data and generally ensures the quality of the output, this component simply produces the raw output from the underlying script. For example, this means that the data produced will have lost and additional properties found in screening file. As such, this component can be thought of as being equivalent to the execution of the script from the command line.



The *Run Virtual Screening* component expects to have a file containing the screening records (**Screening Set** parameter) and another containing the query records (**Query Set** parameter), and exposes parameters which map to arguments for the script. It performs basic validation on these - checking that files are a supported format (by checking for file extensions of .sdf or .mol2), if the files exist, and checking that the content of these files matches the extension (e.g. the .sdf file is a genuine SD file).

In terms of data handling, the component will join the scores produced by the MOL2 file containing the structures screened.

Run Virtual Screening Validation

This component is a low-level wrapper over the python script which performs the screening validation operation. The *Run Virtual Screening Validation* component performs the virtual screening validation operation found in the *Perform Virtual Screening Validation* component. As with the *Run Virtual Screening* component, it is designed to simply verify the input for the script (by checking file formats, if they are found), run the script, then join the data produced (MOL2 and CSV files) together. The *Run Virtual Screening Validation* component expects to have a file containing the query records (**Query Set** parameter), the active records (**Active Set** parameter) and another containing the decoy records (**Decoys Set** parameter).

Run Generate Conformers

This component takes a file (either MOL2 or SDF) and generates conformers for each record found in that file. The number of conformers is limited by the **Max Number of Conformers** parameter. The component can work in multiple ways, according to the parameters.

The most basic operation, the default one, is to simply stream out the result of the conformer generation for the input file. The python script generates two outputs. The first is an SD file containing the molecules and the second file is simply a summary which details how many conformations were found for each molecule, and information about the conformers:

In addition to this, the component can generate a single file (by default) or single files containing all the conformers for a given molecule. In this case, the file name reflects the name of the molecule to which the conformers belong. To generate the files, the **Output Directory** is specified. The files will be written to this location, and the molecule can be merged (**Split Output** parameter set to *"False"*) or split (**Split Output** parameter set to "*True"*).

Derive Script Path

The CSD PP Component Collection package has a script folder which contains the python scripts used to execute on the CSD. To avoid having to reference this folder every time, there is a the *Derive Script Path* component, which knows where this folder is and will gather the path to the script named in its **Script Filename** parameter.

This component will do more than this though. It may be that different instance of CSD require a different script, due to the CSD Python API changes. Therefore, it will examine the CSD instance and determine the best script to use. To do this, scripts must have the version for which they are appropriate in the filename:

`script.1.2.3.py`

Rather than have to second guess the version of the file to use, or know the versions available, the script can be referred to as:

`script.py`

however, it will point to `script.1.2.3.py`

This component will examine all versions of that script, either in the Python API package (if **Script Folder** parameter is blank) or the folder specified in the **Script Folder** parameter and finds the most recent version which is appropriate for this script. For example, if a CSD 0.8.0 version of the script is found, then an earlier version of the script may be used if 0.8.0 is not found (e.g. CSD 0.7.0).

If a single version of the script is not found, this is assumed to mean that `script.py` is appropriate for all versions of CSD and will therefore be used.

Throw Script Error Message

The *Throw Script Error Message* components is a helper component to parse script errors. In particular, this component attempts to find Runtime errors which may relate to licensing and displays those in a better formatted manner - where the Runtime error is positioned at the top of the message to make it clearer.

Check Journal Name

The *Check Journal Name* component will check a journal title against the CSD. This utility component performs a text numeric search on the journal name provided in the **Journal Name** parameter.

Validate Journal Name

The *Validate Journal Name* component checks the validity of a journal title against the CSD. This utility component performs a text numeric search on the journal and returns a *True* or *False* value.

Gather Database Names

The *Gather Database Names* component gather names list and split them into array. It checks for each value for being not empty and additionally that the associated file exists. If all the conditions are satisfied than the name goes forward to the return value.

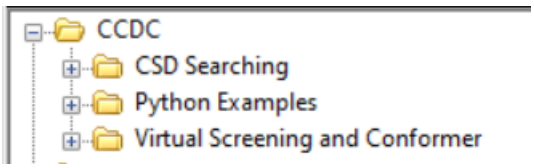
Join Data from JSON

The *Join Data from JSON* component joins a JSON-encoded stream of data that can be specified in the **Source** parameter. The source can be a file, a network resource, a data record property or a global property. The name of the parameter to join can be specified in the **JoinUsing** parameter.

3 CSD Python API Protocols

The CSD PP Component Collection package is supplied with examples designed to demonstrate the use of the different components. No examples are supplied to demonstrate the use of the utilities components, though they are fully tested and their usage can be determined from the examples.

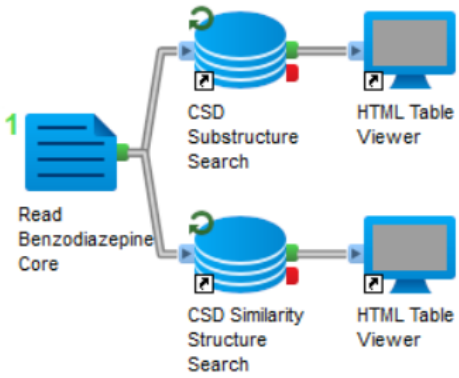
The CSD Python API Protocols are listed under the **CCDC** folder in the **Protocols** tab. The provided protocols are organised in groups based on their purpose: **CSD Searching**, **Python Examples** and **Virtual Screening and Conformer.**



3.1 CSD Searching

The **CSD Searching** protocol folder includes seven different ways to search and to combine searches in the CSD using the CSD Python API.

01 Search CSD By Structure



This protocol uses the benzodiazepine core structure, shipped as part of the Chemistry collection (`data\Queries\BenzodiazepineCore.mol`), and searches the CSD for structures which contain this as a substructure (top) or as a similar compound (bottom). The query file is specified in the **Source** parameter of the *Read Benzodiazepine Core* component. The filters of the search are specified in the **Options** parameter of the *CSD Substructure Search* and *CSD Similarity Structure Search* components**.** The two searches outputs the results in two HTML pages.

The *Substructures* HTML page will provide a table containing as a minimum the refcode(s) for hit structures.

Display records 1 to 25 of 142 Update <<First <Previous Next> Last>>

Substructures		
csd_refcode	database	Name
ANIXAX	as540be_ASER	BenzodiazepineCore
AZEGIX	as540be_ASER	BenzodiazepineCore
AZEGOD	as540be_ASER	BenzodiazepineCore
AZEGUJ	as540be_ASER	BenzodiazepineCore
BATDIL	as540be_ASER	BenzodiazepineCore
BAYCUZ	as540be_ASER	BenzodiazepineCore
BAZCLH	as540be_ASER	BenzodiazepineCore
BCHBZP	as540be_ASER	BenzodiazepineCore
BCHBZP01	as540be_ASER	BenzodiazepineCore
BEDZPN10	as540be_ASER	BenzodiazepineCore
BIXBOZ10	as540be_ASER	BenzodiazepineCore
BIZSAE	as540be_ASER	BenzodiazepineCore
BOFDEF	as540be_ASER	BenzodiazepineCore
BOMMUL	as540be_ASER	BenzodiazepineCore
BORXEL	as540be_ASER	BenzodiazepineCore
BUVLEK	as540be_ASER	BenzodiazepineCore
BZPCUC	as540be_ASER	BenzodiazepineCore
CAGWUC	as540be_ASER	BenzodiazepineCore
CASKIT	as540be_ASER	BenzodiazepineCore
CHABZN	as540be_ASER	BenzodiazepineCore
CICSEM	as540be_ASER	BenzodiazepineCore
CLDZPA	as540be_ASER	BenzodiazepineCore
CLDZPA01	as540be_ASER	BenzodiazepineCore
CLDZPB	as540be_ASER	BenzodiazepineCore
COWHIG	as540be_ASER	BenzodiazepineCore

Display records 1 to 25 of 142 Update <<First <Previous Next> Last>>

While similarity search, in addition, take into account the similarity threshold - which all hit structures must exceed. The similarity threshold is specified in the **Similarity Threshold** parameter. To aid in the interpretation of the results, the *CSD Similarity Structure Search* component outputs an additional property which is the similarity value for this compound to the query for which it was found.

Display records 1 to 25 of 42 Update <<First <Previous Next> Last>>

Similar			
csd_refcode	database	similarity	Name
DCDAZP	as540be_ASER	0.986187845304	BenzodiazepineCore
BODSOC	as540be_ASER	0.87675070028	BenzodiazepineCore
DOZKEI	as540be_ASER	0.832167832168	BenzodiazepineCore
DIZPAM10	as540be_ASER	0.822580645161	BenzodiazepineCore
DIZPAM11	as540be_ASER	0.822580645161	BenzodiazepineCore
ZZZBNV	as540be_ASER	0.822580645161	BenzodiazepineCore
LUTGIR	as540be_ASER	0.81880733945	BenzodiazepineCore
BAZCLH	as540be_ASER	0.816933638444	BenzodiazepineCore
RUMWAZ	as540be_ASER	0.813211845103	BenzodiazepineCore
VERZEX	as540be_ASER	0.78982300885	BenzodiazepineCore
CERBEG	as540be_ASER	0.780193236715	BenzodiazepineCore
CERBEG01	as540be_ASER	0.780193236715	BenzodiazepineCore
BORXEL	as540be_ASER	0.774403470716	BenzodiazepineCore
SUXFUM	as540be_ASER	0.774403470716	BenzodiazepineCore
SUXFOG	as540be_ASER	0.771058315335	BenzodiazepineCore
FULWUE	as540be_ASER	0.762820512821	BenzodiazepineCore
LUTGAJ	as540be_ASER	0.757961783439	BenzodiazepineCore
NHPBZO	as540be_ASER	0.757961783439	BenzodiazepineCore
QUGGUU	as540be_ASER	0.757961783439	BenzodiazepineCore
QUGHAB	as540be_ASER	0.757961783439	BenzodiazepineCore
FLDAZP	as540be_ASER	0.753164556962	BenzodiazepineCore
OXAPAM10	as540be_ASER	0.742203742204	BenzodiazepineCore
VICFIW	as540be_ASER	0.734567901235	BenzodiazepineCore
ZZZAUS10	as540be_ASER	0.734567901235	BenzodiazepineCore
ZZZAUS20	as540be_ASER	0.734567901235	BenzodiazepineCore

Display records 1 to 25 of 42 Update <<First <Previous Next> Last>>

02 Search CSD By Text Numeric Fields

This protocol shows how the CSD can be searched using text and numeric fields. In this particular case, two text and numeric searches are performed.

The first *CSD Text Numeric Search* looks for all crystals reported in *"Acta Crystallogr.,Sect.B:Struct.Crystallogr.Cryst.Chem"* as specified in the **Citations** parameter. This is limited to 100 records by setting the **Maximum** parameter.

Display records 1 to 25 of 100 Update <<First <Previous Next> Last>>

First 100 for Acta Crystallogr.,Sect.B:Struct.Crystallogr.Cryst.Chem.

csd_refcode	publications	database
AADAMC	(Citation(authors=u'K.K.Chacko, R.Zand', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'29', year=1973, first_page=u'2681', doi=u'10.1107/S0567740873007363'),)	as540be_ASER
AAMTXP	(Citation(authors=u'M.W.Wieczorek, W.S.Sheldrick, J.Karolak-Wojciechowska, M.Mikolajczyk, B.Ziemnicka', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'35', year=1979, first_page=u'2339', doi=u'10.1107/S0567740879009250'),)	as540be_ASER
AANHOX	(Citation(authors=u'M.R.Ciajolo, F.Lelj, T.Tancredi, P.A.Temussi', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'37', year=1981, first_page=u'1130', doi=u'10.1107/S0567740881005256'),)	as540be_ASER
AAXTHP	(Citation(authors=u'B.Kojic-Prodic, V.Rogic, Z.Ruzic-Toros', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'32', year=1976, first_page=u'1833', doi=u'10.1107/S0567740876006481'),)	as540be_ASER
ABBUMO10	(Citation(authors=u'F.R.Ahmed', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'37', year=1981, first_page=u'188', doi=u'10.1107/S0567740881002471'),)	as540be_ASER
ABCLUA10	(Citation(authors=u'A.Ducruix, C.Pascard, S.Omura, A.Nakagawa', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'33', year=1977, first_page=u'2314', doi=u'10.1107/S0567740877008334'),)	as540be_ASER
ABCNCX	(Citation(authors=u'S.K.Bhattacharjee, K.K.Chacko, R.Zand', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'35', year=1979, first_page=u'399', doi=u'10.1107/S0567740879003629'),)	as540be_ASER
ABCOCX	(Citation(authors=u'K.K.Chacko, S.K.Bhattacharjee, R.Zand, R.Water', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'34', year=1978, first_page=u'147', doi=u'10.1107/S0567740878002496'),)	as540be_ASER
ABDECO10	(Citation(authors=u'R.Ramachandra, K.Vijayan', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'29', year=1973, first_page=u'1945', doi=u'10.1107/S0567740873005807'),)	as540be_ASER
ABFERC10	(Citation(authors=u'G.Calvarin, D.Weigel', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'27', year=1971, first_page=u'1253', doi=u'10.1107/S0567740871003820'),)	as540be_ASER
ABGPON	(Citation(authors=u'A.Mercer, J.Trotter', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'34', year=1978, first_page=u'1596', doi=u'10.1107/S0567740878006123'),)	as540be_ASER
ABINOR01	(Citation(authors=u'S.Takagi, S.Nordenson, G.A.Jeffrey', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'35', year=1979, first_page=u'991', doi=u'10.1107/S0567740879005379'),)	as540be_ASER
ABINOS01	(Citation(authors=u'S.Takagi, G.A.Jeffrey', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'33', year=1977, first_page=u'3033', doi=u'10.1107/S0567740877010218'),)	as540be_ASER
ABINOS02	(Citation(authors=u'G.A.Jeffrey, A.Robbins, R.K.McMullan, S.Takagi', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'36', year=1980, first_page=u'373', doi=u'10.1107/S0567740880003299'),)	as540be_ASER
ABIPIM	(Citation(authors=u'P.Sommerville, M.Laing', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'32', year=1976, first_page=u'2683', doi=u'10.1107/S0567740876008534'),)	as540be_ASER
ABMQZD	(Citation(authors=u'M.H.P.Ardebili, J.G.White', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'34', year=1978, first_page=u'2890', doi=u'10.1107/S0567740878009528'),)	as540be_ASER
ABNCAM	(Citation(authors=u'G.L.Dwivedi, R.C.Srivastava', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'28', year=1972, first_page=u'2567', doi=u'10.1107/S056774087206648X'),)	as540be_ASER
ABORCR10	(Citation(authors=u'I.F.Taylor Junior, E.A.H.Griffith, E.L.Amma', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'32', year=1976, first_page=u'653', doi=u'10.1107/S0567740876003695'),)	as540be_ASER
ABPZOL10	(Citation(authors=u'J.Lapasset, A.Escande, J.Falgueirettes', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'28', year=1972, first_page=u'3316', doi=u'10.1107/S0567740872007897'),)	as540be_ASER
ABRBPH	(Citation(authors=u'H.H.Sutherland, T.G.Hoy', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'25', year=1969, first_page=u'2385', doi=u'10.1107/S0567740869005759'),)	as540be_ASER
ABRTOL	(Citation(authors=u'H.van der Meer', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'28', year=1972, first_page=u'3098', doi=u'10.1107/S0567740872007484'),)	as540be_ASER
ABSCIC	(Citation(authors=u'P.Swaminathan, J.Vijayalakshmi, R.Srinivasan', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'32', year=1976, first_page=u'2351', doi=u'10.1107/S0567740876007747'),)	as540be_ASER
ABSFCN	(Citation(authors=u'K.K.Kakati, B.Chaudhuri', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'24', year=1968, first_page=u'1645', doi=u'10.1107/S0567740868004796'),)	as540be_ASER
ABTBTR	(Citation(authors=u'J.Galloy, J.P.Declercq, M.van Meerssche', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'34', year=1978, first_page=u'974', doi=u'10.1107/S0567740878004550'),)	as540be_ASER
ABTNBA	(Citation(authors=u'I.Bar, J.Bernstein', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'37', year=1981, first_page=u'569', doi=u'10.1107/S0567740881003592'),)	as540be_ASER

Display records 1 to 25 of 100 Update <<First <Previous Next> Last>>

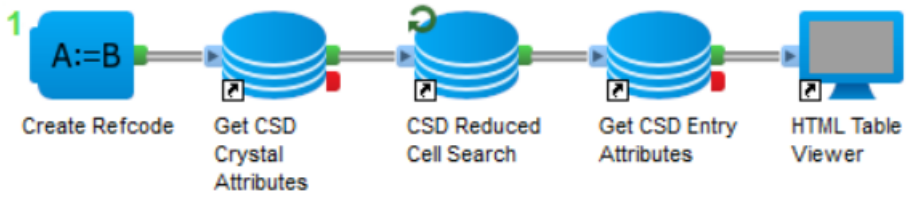
The second *CSD Text Numeric Search* component also looks in that journal, but now also limits the hits returned to only those were the colour of the crystal is *"orange"*. This filter is specified in the **Color** parameter of the second *CSD Text Numeric Search* component. Note here that the maximum is still set, so we still get 100 hits. However, this second search, we only get crystals that are orange, or contain the word orange (e.g. red-orange).

Display records 1 to 25 of 100 Update <<First <Previous Next> Last>>

Orange Crystals in Acta Crystallogr.,Sect.B:Struct.Crystallogr.Cryst.Chem.		
csd_refcode	color	database
ABTNBA	orange	as540be_ASER
ACDXUS	light orange	as540be_ASER
ACPSMO	orange	as540be_ASER
ACRMSA	orange	as540be_ASER
ACRMSB	red-orange	as540be_ASER
AMALCO10	orange	as540be_ASER
AMAZBR	orange	as540be_ASER
AMCQUN	orange	as540be_ASER
AMGZCO	dark orange-red	as540be_ASER
AZBNOS	red-orange	as540be_ASER
AZDETO10	orange	as540be_ASER
BADVIL10	yellow-orange	as540be_ASER
BAFXEL	red-orange	as540be_ASER
BAFXUB	orange-red	as540be_ASER
BALACF	orange-red	as540be_ASER
BATCAA01	orange-yellow	as540be_ASER
BAZBIN	orange-red	as540be_ASER
BDMIUP	brown orange	as540be_ASER
BECVEK	red-orange	as540be_ASER
BEJRUD	orange-red	as540be_ASER
BEJSUE	yellow-orange	as540be_ASER
BEPNAL	red-orange	as540be_ASER
BERMIU	yellow-orange	as540be_ASER
BEWKAP	dark orange	as540be_ASER
BEYNEY	pale orange	as540be_ASER

Display records 1 to 25 of 100 Update <<First <Previous Next> Last>>

03 Search CSD By Reduced Cell



This example starts with a Refcode that is specified in the **Expression** parameter of the *Create Refcode* component. For this Refcode, crystal structure details are retrieved, cell lengths, cell angles, and lattice centring information that are specified in the **Attributes < CSD RefCode Property** parameter of the *Get CSD Crystal Attributes* component. These details are then used in a reduced cell search.

Reduced cell searches can be carried out in two ways.

In this example, the cell lengths and angles returned by the crystal attributes are used. These are strings of the form:

```
CellLengths(a=8.4708, b=10.0492, c=14.0363)
CellAngles(alpha=86.016, beta=79.914, gamma=71.818)
```

The alternative is to enter the `a, b, c`, and `alpha, beta, gamma` values separately in the appropriate parameters of the *CSD Reduced Cell Search* component.

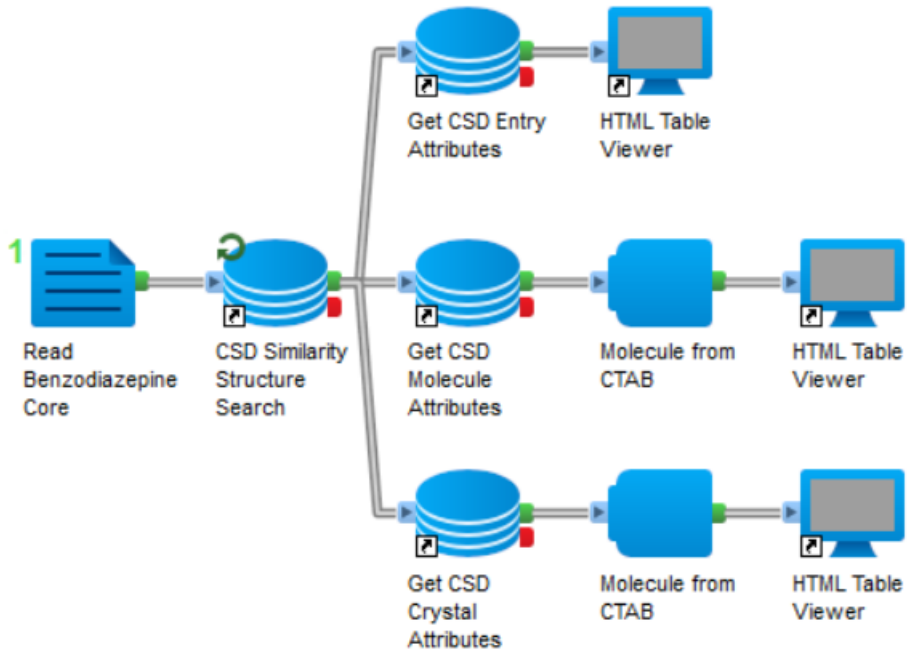
With the refcodes found by this search, the chemical name is retrieved, and the results reported in a HTML table.

Display records 1 to 25 of 27 Update <<First <Previous Next> Last>>

csd_refcode	chemical_name	database
AACMHX10	-Acetoxy-,2-anti-diphenylmethylene-cyclohexane	as540be_ASER
BIGRIT	bis(2-Pyrrolidino)-tetramethyl-di-indium(iii)	as540be_ASER
BOLCUA01	Mesitil	as540be_ASER
BOLCUA11	2,2',4,4',6,6'-Hexamethylbenzil	as540be_ASER
FIDMIR	bis(3,4,7,8-tetrahydro-2H,6H-pyrimido[1,2-a]pyrimidinyl)bis(chloro)diboron	as540be_ASER
GEBYIV	bis((2-Dimethyl phosphonato-P,O)-dicarbonyl-(trifluoroacetato-O,O')-(trimethylphosphite-P)-molybdenum)	as540be_ASER
JENMOE	bis(5-Cyclopentadienyl)-(2-6,6-1,2,3,4-tetramethyl-1,4-dibora-2,5-cyclohexadienyl)-di-nickel	as540be_ASER
MONZAR	4-(4,5-Diphenyl-1H-imidazol-2-yl)benzonitrile	as540be_ASER
PCYPCR	(2.2)Paracyclophane bis-O-18-crown-6 ether	as540be_ASER
PCYPCR01	4,5,15,16-bis(18-Crown-6)(2.2)paracyclophane	as540be_ASER
PCYPCR10	4,5,15,16-bis(18-Crown-6)(2.2)paracyclophane	as540be_ASER
QOBPUT	(R,R)-(6-1-(1-Pyrrolidinyl)-2-(1-(dimethylamino)ethyl)benzene)-tricarbonyl-chromium	as540be_ASER
RABNIT	catena-[tetrakis(benzimidazole)-zinc(ii) tris(2-oxalato)-di-zinc(ii)]	as540be_ASER
REDJUF	catena-tris((2-5-Cyclopentadienyl)-(5-cyclopentadienyl)-lead) toluene solvate	as540be_ASER
TEJFUM	(S,Z)-5-chloro-2-fluoro-4-(((tetrahydro-2H-pyran-3-yl)methyl)amino)-N-(thiazol-2(3H)-ylidene)benzenesulfonamide	as540be_ASER
VEZSOI	Chloro-(6-hexamethylbenzene)-trimethylphosphine-(5-trimethylsiloxy-5,5-diphenylpent-1,3-diynyl)-ruthenium	as540be_ASER
VOXDIV	(R,S)-3,3,6,6-Tetramethyl-1,4-diphenyl-2,5,7-trioxabicyclo(2.2.1)heptane	as540be_ASER
WIRNOC	(2,6-bis(3,5-bis(trifluoromethyl)phenyl)-4-phenylphosphinine)-(5-pentamethylcyclopentadienyl)-chloro-iridium	as540be_ASER
WOPKER	(6-2,2,2,2-Buta-1,3-diyne-1,4-diyl)-(2-hydrido)-heptadecacarbonyl-(5-cyclopentadienyl)-penta-ruthenium-tungsten	as540be_ASER
WOPKOB	(6-2,2,2,2-Buta-1,3-diyne-1,4-diyl)-(2-hydrido)-heptadecacarbonyl-(5-cyclopentadienyl)-iron-tetra-ruthenium-tungsten	as540be_ASER
WOPKOB01	(6-2,2,2,2-Buta-1,3-diyne-1,4-diyl)-(2-hydrido)-heptadecacarbonyl-(5-cyclopentadienyl)-iron-tetra-ruthenium-tungsten	as540be_ASER
WOPLAO	(6-2,2,2,2-Buta-1,3-diyne-1,4-diyl)-(2-hydrido)-heptadecacarbonyl-(5-cyclopentadienyl)-di-iron-tri-ruthenium-tungsten	as540be_ASER
WOPLAO01	(6-2,2,2,2-Buta-1,3-diyne-1,4-diyl)-(2-hydrido)-heptadecacarbonyl-(5-cyclopentadienyl)-di-iron-tri-ruthenium-tungsten	as540be_ASER
WOTREC	(2-4,6-Diacetylresorcinolato)-bis(1,10-phenanthroline)-di-copper(ii) bis(tetrafluoroborate)	as540be_ASER
XATPOX	N-Methyl-N-(4-methyl-6-phenyl-2H-pyran-2-ylidene)-N-phenylammonium iodide	as540be_ASER

Display records 1 to 25 of 27 Update <<First <Previous Next> Last>>

04 Retrieve Entry and Molecule Attributes



This protocol uses the benzodiazepine core structure, shipped as part of the Chemistry collection (`data\Queries\BenzodiazepineCore.mol`), and searches the CSD for structures which contain this as a similar compound. The query file is specified in the **Source** parameter of the *Read Benzodiazepine Core* component.

The *CSD Similarity Search* components return refcodes only. To gather details about that refcode, the *Get CSD Entry Attributes*, the *Get CSD Molecule Attributes* and the *Get CSD Crystal Attributes* components can be used.

A CSD entry contains attributes that are beyond the concepts of chemistry and crystallography. An example of such an attribute would be the publication details of a CSD entry.

The CSD molecule represents the chemistry and chemical attributes associated with an entry such as the structure, the molecular weight, etc. The structure is found in the following forms: SMILES, CTAB and MOL2.

In some cases, a molecule may not have a canonical SMILES representation (e.g when the structure has unknown atoms or bonds). In these cases, the value will be None. An example would be `AJABIX01`, the SMILES string property is removed to avoid confusion.

This protocol returns three HTML reports, one for the *Entry Attributes*, one for the *Molecule Attributes* and one for the *Crystal Attributes* of the CSD entries similar to the query.

05 Combining Hit Sets - AND

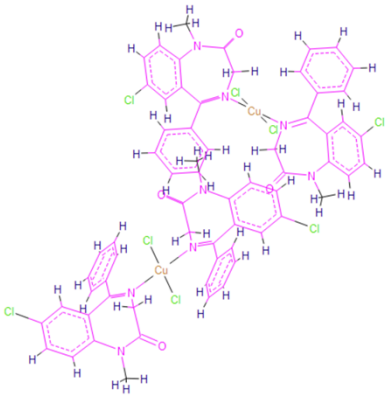
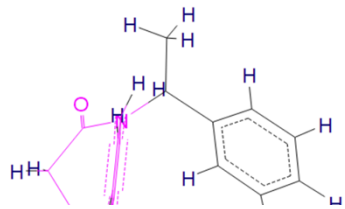
This protocol demonstrates how various queries can be combined, by performing them separately and then applying logic to the list of refcodes produced.

In this case, we perform a substructure search for benzodiazepine core (shipped as part of the Chemistry collection (`data\Queries\BenzodiazepineCore.mol`), which leads to one stream of CSD refcodes. Then, we tag this stream such that each record receives an arbitrary property *"A"* - just so we can identify these refcodes downstream- defined in the **TagName** parameter.

We then perform a search by citation for all crystals in *"Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem"*, using the appropriate abbreviation in the **Citation** parameter. This produces a second stream of refcodes, which then we tag with an arbitrary property of *"B"*.

We merge these streams using the refcodes. A filter is then applied to the resultant list of refcodes; if both the *"A"* and *"B"* properties are found, then this must have come from both the CSD substructure search and journal search and therefore this passes the filter. Otherwise, the record is passed to the fail port.

With the passing data, we collect both the publication information (specified in the **Attributes** parameter of the *Get CSD Entry Attributes* component) and the structure information, to demonstrate that the journal is indeed as we had asked for, and the structure contains a benzodiazapine core. The structure is highlighted using the original query to emphasise this point and results reported in an HTML page.

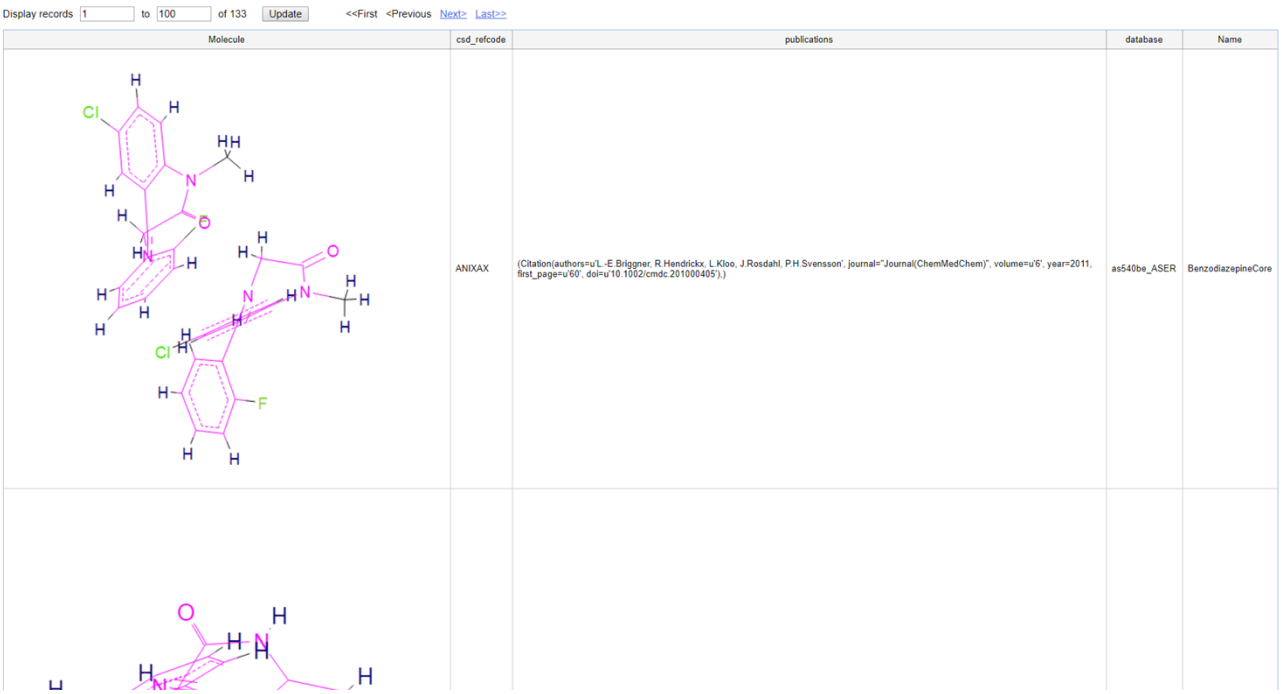
Molecule	csd_refcode	publications	database	Name
	BAYCUZ	(Citation(authors=u'H.Miyamae, A.Obata, H.Kawazura', journal="Journal(Acta Crystallographica,Section B:Struct.Crystallogr.Cryst.Chem. [1968-1982])", volume=u'38', year=1982, first_page=u'272', doi=u'10.1107/S0567740882002593',))	as540be_ASER	BenzodiazepineCore
	BEDZPN10	(Citation(authors=u'Z.Ruzic-Toros, B.Kojic-Prodic, N.Bresciani-Pahor, G.Nardin, L.Randaccio', journal="Journal(Acta Crystallographica,Section	as540be_ASER	BenzodiazepineCore

06 Combining Hit Sets – NOT

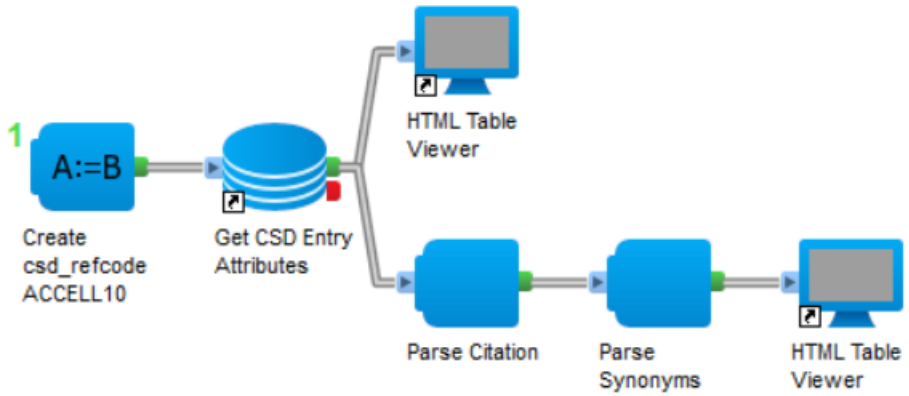
This protocol takes the same premise as for *05 Combining Hit Sets - AND*, and queries for both a structure and a journal.

This time the stream of refcodes produced is combined such that the data is passed only if the structure search found it. This means that if the journal search found the data, then the record is failed. This filter is specified in the **Expression** parameter of the *Is From Set A Note Set B?* component.

As before, the stream data produced are tagged. This time the logic is to pass if tag *"A"* is defined and tag "*B"* is not defined. The appropriate attributes (such as structure and journal) are retrieved, and the substructure is highlighted in the structures before reporting.



07 Parsing Citation and Synonyms



This example shows how the data returned by the Python API can be further enhanced.

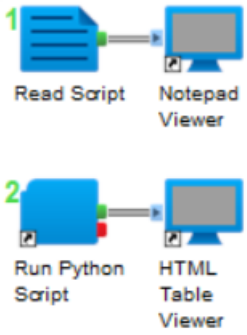
In this case, we pass a refcode (`ACCELL10`) and take the publication and synonyms in the citations. These are specified in the **Attribute** parameter of the component. Then we produce separate properties for each field in the citation (publications and synonyms in this case). We take the synonyms and convert them into arrays. In both cases, the encoding used by python is stripped away also.

Finally, we convert the Unicode found in the synonyms into their appropriate characters (e.g. \u03b2 becomes a "beta" character **β**).

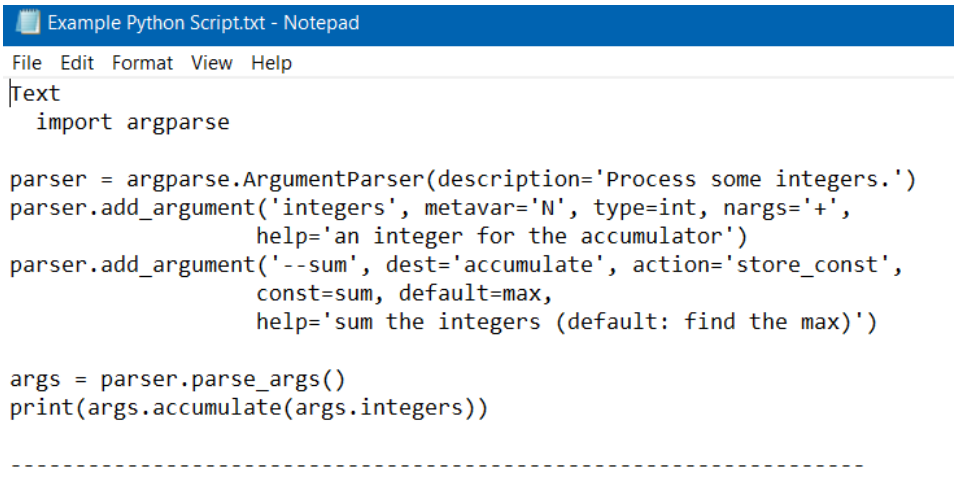
3.2 Python Examples

This group of protocols show few examples of how using CSD Python API.

01 Run Python Script Example



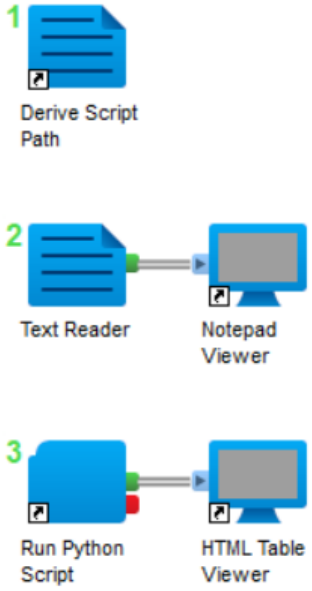
This protocol takes a simple example script shipped as part of the CSD PP Component Collection (`data\Example Scripts\example.py`) and displays it to the user before executing it. The script is simple, taking a set of numeric inputs and summing them together. The script is specified in the **Source** parameter of the *Read Script* component.



Switching the final argument to `--max` will return the max value instead.

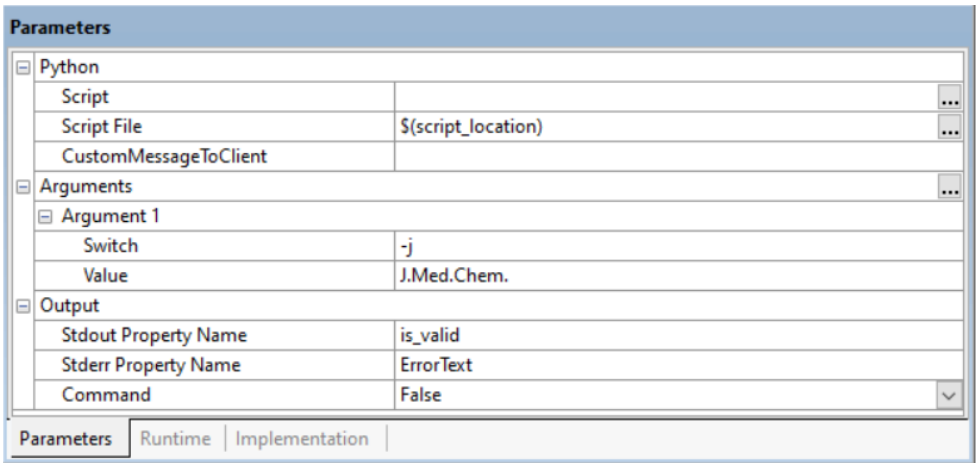
It demonstrates how the Run Python component can be configured to point to any Python script and execute it.

02 Using Derive Script Path



The CSD PP Component Collection package has a script folder which contains the python scripts used to execute on the CSD. To avoid having to reference this folder every time, there is a the *Derive Script Path* component, which knows where this folder is and will gather the path to the script named in its **Script Filename** parameter.

In this protocol, the **Script Filename** points to a simple script named `validate_journal.py`. The script is displayed to the user before executing it. The script takes precisely one argument; "*-j*" for the journal name to test. The arguments for the script are specified in the **Arguments** parameter of the *Run Python Script* component of the protocol.

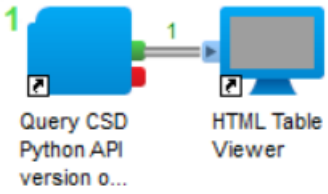


The python script produces two properties; as specified in the **Output** parameters:

Stdout Property Name named *"is_valid"* which will contain "*True*" if the journal name is valid, else it will return "*False*" and **Stderr Property Name** named *"ErrorText".* If there are errors in the execution of the script, these will appear in the "*ErrorText"* property. The results of the script will be reported in an HTML page.

is_valid	ErrorText
True	

03 Get CSD Python API Version

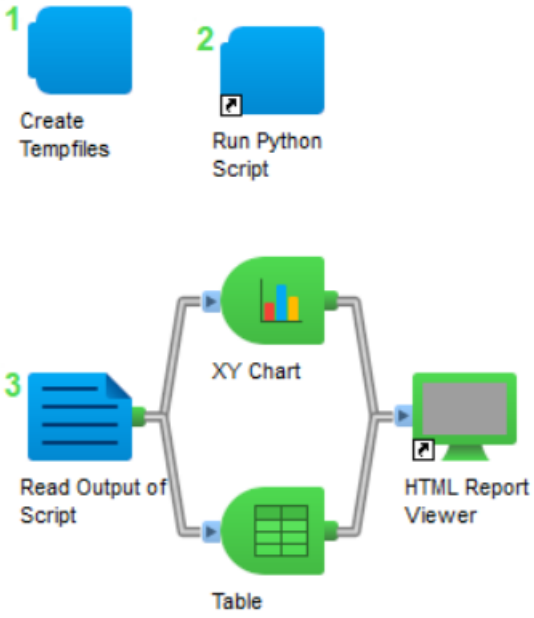


This protocol runs a python script on the server to determine the installed CSD Python API version and the installed python interpreter.

The script is defined in the **Script** parameter of the *Query CSD Python API version on server* component. The script produces two properties; as specified in the **Output** parameters: **Stdout Property Name** named *"script_out"* which will contain the CSD Python API version and the python interpreter and **Stderr Property Name** named *"script_err".* If there are errors in the execution of the script, these will appear in the *script_err* property. The results of the script will be reported in an HTML page.

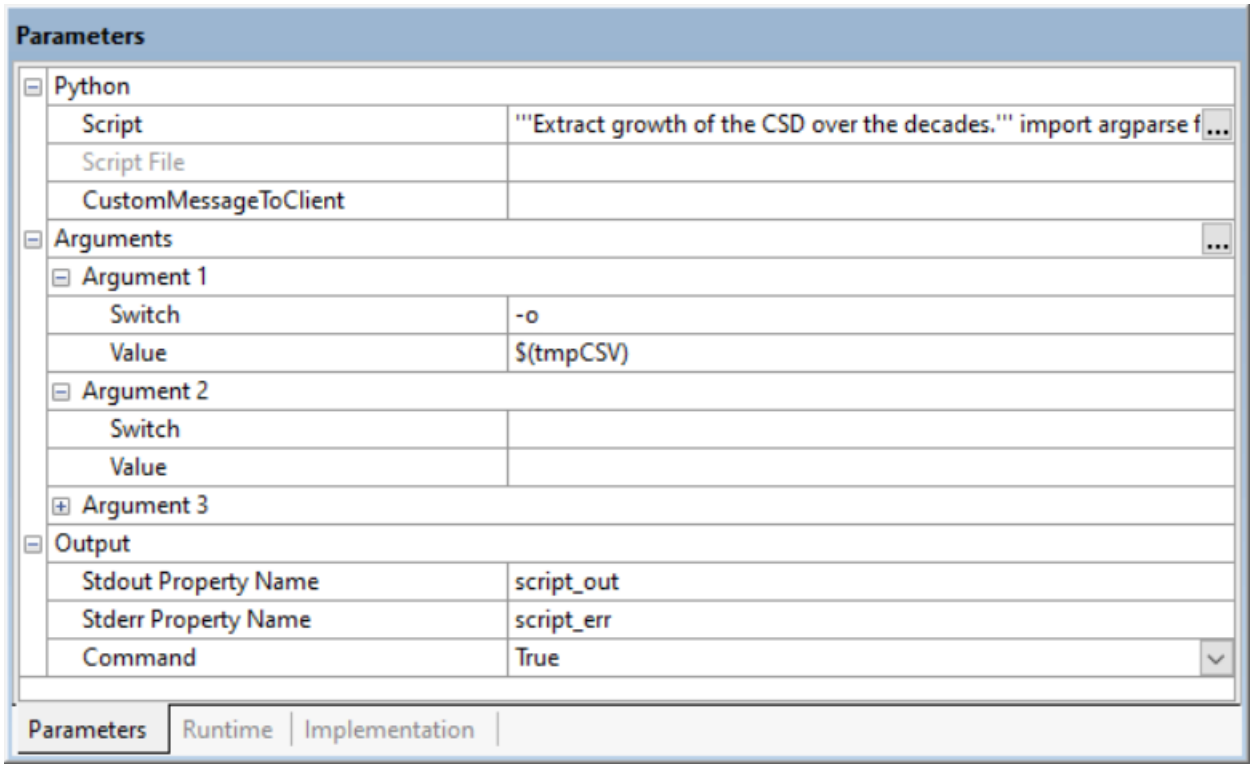
CSD Python API version	
script_out	script_err
2.2.0 sys.version_info(major=2, minor=7, micro=15, releaselevel='final', serial=0)	

04 Count Entries per Decade



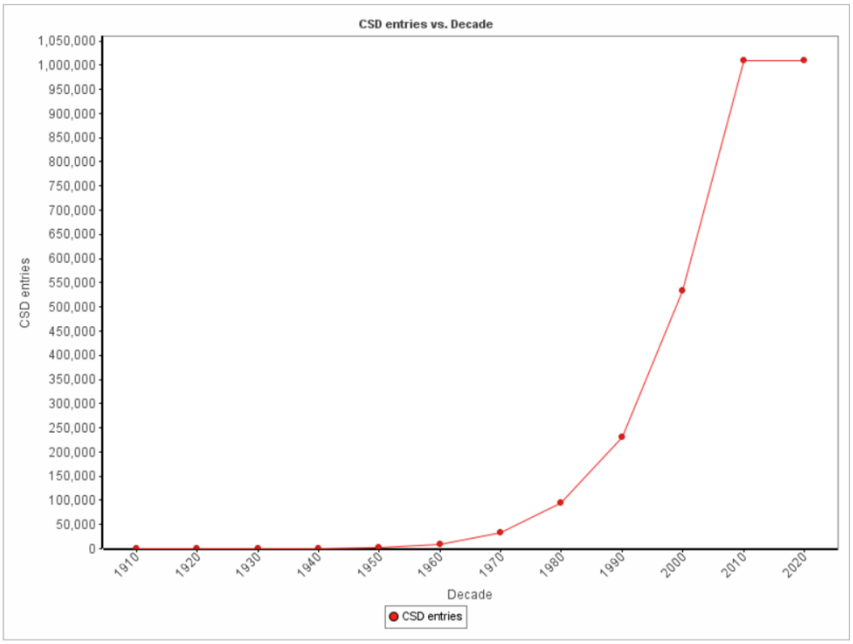
This protocol shows how to implement a more complex script, one which uses the CSD Python API.

It starts by creating a global variable to contain the path to a temporary file(*"tmpCSV")*. Then the script, specified in the **Script** parameter, is passed this path in an argument, *"-o*", so the script may use this path.



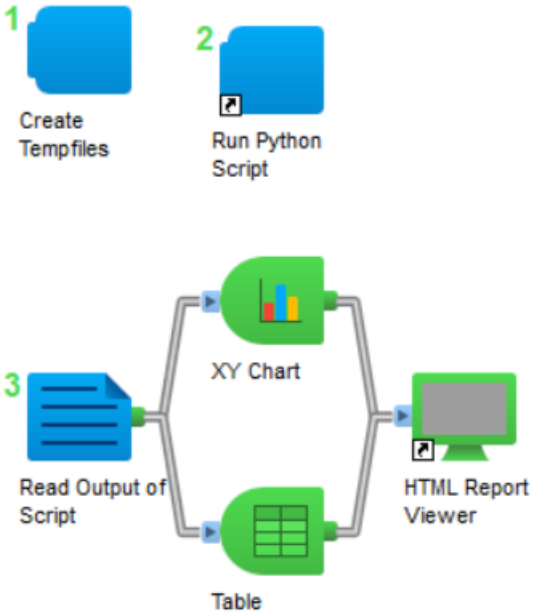
The script populates the file with the breakdown of entries in CSD by decade of their reporting in the literature.

The protocol then picks this temporary file up and reports on it generating an HTML page with the plot of the CSD entries per decade and a table with these values.



Decade	CSD entries
1910	0
1920	64
1930	496
1940	909
1950	2314
1960	8398
1970	32626

05 Count Entries per Year



As the *04 Count Entries per Decade* protocol, this example shows how to implement a more complex script, one which uses the CSD Python API.

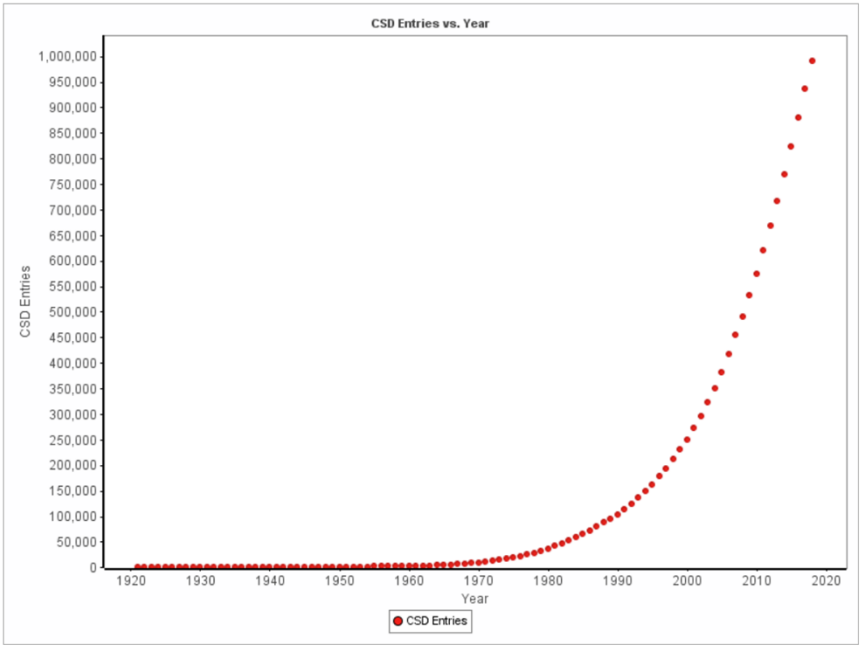
It starts by creating a global variable to contain the path to a temporary file(*"tmpCSV")*. Then, the script specified in the **Script** parameter passes this path to the temporary file in an argument, *"-o"*, so the script may use this path.

Python	
Script	'''Extract growth of the CSD over the years.''' import argparse ...
Script File	
CustomMessageToClient	
Arguments	...
Argument 1	
Switch	-o
Value	$(tmpCSV)
Argument 2	
Switch	
Value	
Argument 3	
Output	
Stdout Property Name	script_out
Stderr Property Name	script_err
Command	True

Parameters | Runtime | Implementation

The script populates the file with the breakdown of entries in CSD by year of their reporting in the literature.

The protocol then picks this temporary file up and reports on it generating an HTML page with the plot of the number of CSD entries per year and a table with these values.

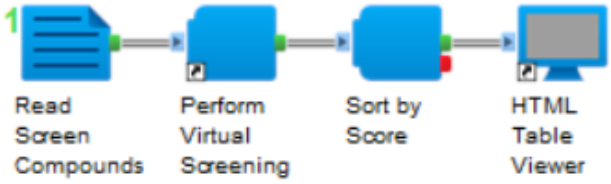


Year	CSD entries
1921	0
1922	0
1923	2
1924	3
1925	8
1926	13
1927	26

3.3 Virtual Screening and Conformer

These protocols are available for user with a CSD-Discovery and/or CSD-Material licence. The twelve protocols provide different workflows on how to perform virtual screening, virtual screening validation and conformer generation using CSD Python API.

01 Queries Identified by File Screening Example



This protocol takes a stream of incoming molecules, and screens against a query set identified by a file in order to generate a virtual screening score.

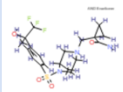
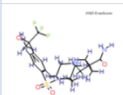
The stream of molecules used in this protocol is provided as part of the CSD PP Component Collection (`data\Python API Example Data\P28845_actives.sdf`).

The query set may be supplied either as a MOL2 or SD file and it is specified in the **Source** parameter of the *Perform Virtual Screening* component. The query used in this protocol is provided as part of the CSD PP Component Collection (`data\Python API Example Data\P28845.sdf`). Up to twenty-five conformations of each molecule are generated and the calculation is distributed over one thread as specified in the **Screening Options** parameters of the *Perform Virtual Screening* component.

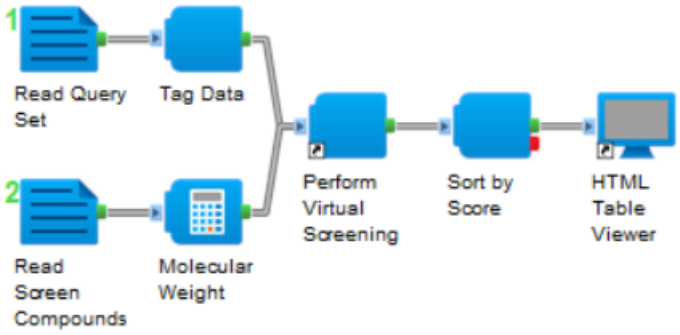
The virtual screening score is calculated and then the incoming data are sorted by screening score from lowest to highest. The sort criteria are defined by one or more properties in the **Sort By**

parameter whose values are used to order the records. The results are then displayed in an HTML page.

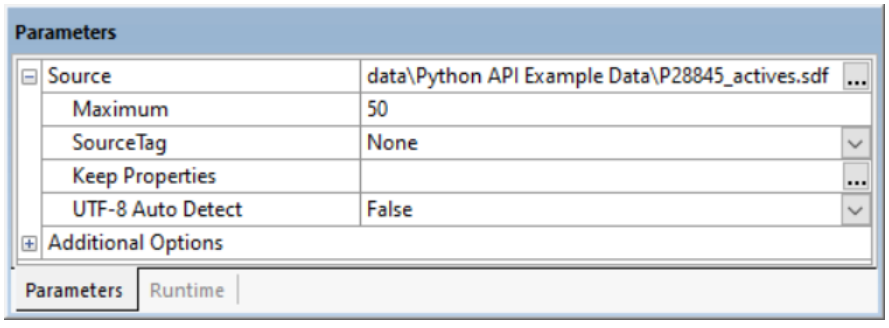
Display records 1 to 25 of 164 Update <<First <Previous Next> Last>>

Data Image	Mol2_MolInfo_Name	Mol2_MolInfo_Type	Mol2_MolInfo_Charge	Mol2_MolInfo_Comment	Mol2_MolInfo_StatusBits	Mol2_Substructures	Name	Virtual_Screen_Score
	CHEMBL460962 P10000288	SMALL	USER_CHARGES	Generated from the CSD	****	1 NONPOLAR 1 GROUP 0 8 NONPOLAR 0 2 unknown 3 GROUP 0 0 unknown 0 3 DONOR_STRONG 5 GROUP 0 4 DONOR_STRONG 0 4 ACCEPTOR_MEDIUM 11 GROUP 0 3 ACCEPTOR_MEDIUM 0 5 DONACC 25 GROUP 0 1 DONACC 0 6 DONOR_MEDIUM 32 GROUP 0 5 DONOR_MEDIUM 0	CHEMBL460962 P10000288	-127.996
	CHEMBL460962 P10000287	SMALL	NO_CHARGES	Generated from the CSD	****	1 NONPOLAR 1 GROUP 0 8 NONPOLAR 0 2 unknown 3 GROUP 0 0 unknown 0 3 ACCEPTOR_MEDIUM 5 GROUP 0 3 ACCEPTOR_MEDIUM 0 4 DONACC 24 GROUP 0 1 DONACC 0 5 DONOR_MEDIUM 31 GROUP 0 5 DONOR_MEDIUM 0	CHEMBL460962 P10000287	-124.099
	CHEMBL221158 P10000194	SMALL	NO_CHARGES	Generated from the CSD	****	1 NONPOLAR 1 GROUP 0 8 NONPOLAR 0 2 unknown 3 GROUP 0 0 unknown 0 3 ACCEPTOR_MEDIUM 11 GROUP 0 3 ACCEPTOR_MEDIUM 0 4 DONOR_MEDIUM 19 GROUP 0 5 DONOR_MEDIUM 0 5 ACCEPTOR_WEAK 28 GROUP 0 6 ACCEPTOR_WEAK 0	CHEMBL221158 P10000194	-114.883
	CHEMBL510937 P10000220	SMALL	USER_CHARGES	Generated from the CSD	****	1 NONPOLAR 1 GROUP 0 8 NONPOLAR 0 2 unknown 3 GROUP 0 0 unknown 0 3 ACCEPTOR_STRONG 11 GROUP 0 7 ACCEPTOR_STRONG 0 4 ACCEPTOR_MEDIUM 13 GROUP 0 3 ACCEPTOR_MEDIUM 0	CHEMBL510937 P10000220	-112.336
	CHEMBL222576 P10000001	SMALL	NO_CHARGES	Generated from the CSD	****	1 NONPOLAR 1 GROUP 0 8 NONPOLAR 0 2 unknown 3 GROUP 0 0 unknown 0 3 ACCEPTOR_MEDIUM 11 GROUP 0 3 ACCEPTOR_MEDIUM 0 4 DONOR_MEDIUM 19 GROUP 0 5 DONOR_MEDIUM 0	CHEMBL222576 P10000001	-111.418
								

02 Queries Identified by Tag Screening Example



This protocol shows the ability to stream in both query and screening molecules, as long as the query molecules are tagged so they may be identified. The tag name is specified in the **TagName** parameter of the *Tag Data* component. The query and the stream of molecule are specified in the **Source** parameter of the *Read* components. For the screening molecules, only the first fifty molecules will be used as specified in the **Maximum** parameter of the *Read Screen Compounds* component.



The molecular weight property is added to the screening set to demonstrate that the data defined on the original dataset is passed through this component.

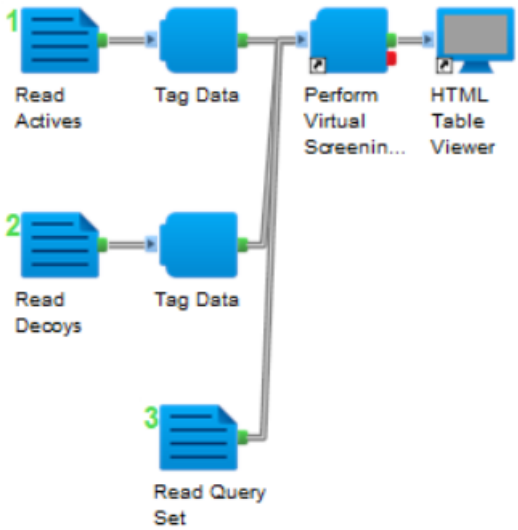
The *Perform Virtual Screening* component then internally divides the records according to the **Test for Query**, "*IsQuery is Defined"* to produce a query et and screening set.

The data, which is output by the component, has a new virtual screening score which is stored in a property, those are then sorted by screening score from lowest to highest. The results are then displayed in an HTML page.

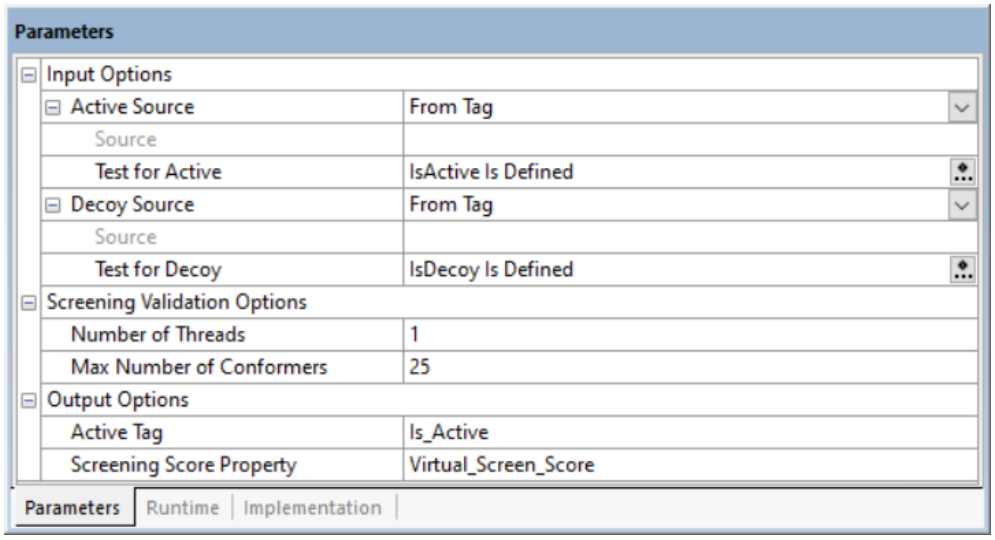
Display records 1 to 25 of 50 Update <<First <Previous Next> Last>>

Data Image	Mol2_MolInfo_Name	Mol2_MolInfo_Type	Mol2_MolInfo_Charge	Mol2_MolInfo_Comment	Mol2_MolInfo_StatusBits	Mol2_Substructures	Name	Virtual_Screen_Score	Molecular_Weight
	CHEMBL510937 P10000220	SMALL	USER_CHARGES	Generated from the CSD	****	1 NONPOLAR 1 GROUP 0 8 NONPOLAR 0 2 unknown 3 GROUP 0 0 unknown 0 3 ACCEPTOR_STRONG 11 GROUP 0 7 ACCEPTOR_STRONG 0 4 ACCEPTOR_MEDIUM 13 GROUP 0 3 ACCEPTOR_MEDIUM 0	CHEMBL510937 P10000220	-113.304	424.51
	CHEMBL401359 P10000034	SMALL	NO_CHARGES	Generated from the CSD	****	1 NONPOLAR 1 GROUP 0 8 NONPOLAR 0 2 ACCEPTOR_MEDIUM 10 GROUP 0 3 ACCEPTOR_MEDIUM 0 3 DONOR_MEDIUM 24 GROUP 0 5 DONOR_MEDIUM 0 4 unknown 25 GROUP 0 0 unknown 0	CHEMBL401359 P10000034	-111.481	349.43
	CHEMBL222576 P10000001	SMALL	NO_CHARGES	Generated from the CSD	****	1 NONPOLAR 1 GROUP 0 8 NONPOLAR 0 2 unknown 3 GROUP 0 0 unknown 0 3 ACCEPTOR_MEDIUM 11 GROUP 0 3 ACCEPTOR_MEDIUM 0 4 DONOR_MEDIUM 19 GROUP 0 5 DONOR_MEDIUM 0	CHEMBL222576 P10000001	-110.916	408.32
	CHEMBL464387 P10000020	SMALL	USER_CHARGES	Generated from the CSD	****	1 NONPOLAR 1 GROUP 0 8 NONPOLAR 0 2 unknown 3 GROUP 0 0 unknown 0 3 ACCEPTOR_STRONG 11 GROUP 0 7 ACCEPTOR_STRONG 0 4 ACCEPTOR_MEDIUM 13 GROUP 0 3 ACCEPTOR_MEDIUM 0	CHEMBL464387 P10000020	-110.745	440.51
	CHEMBL249227 P10000190	SMALL	NO_CHARGES	Generated from the CSD	****	1 NONPOLAR 1 GROUP 0 8 NONPOLAR 0 2 ACCEPTOR_MEDIUM 8 GROUP 0 3 ACCEPTOR_MEDIUM 0 3 unknown 28 GROUP 0 0 unknown 0	CHEMBL249227 P10000190	-110.042	382.48
	CHEMBL399455 P10000029	SMALL	NO_CHARGES	Generated from the CSD	****	1 NONPOLAR 1 GROUP 0 8 NONPOLAR 0 2 ACCEPTOR_MEDIUM 8 GROUP 0 3 ACCEPTOR_MEDIUM 0 3 unknown 28 GROUP 0 0 unknown 0	CHEMBL399455 P10000029	-109.539	364.44

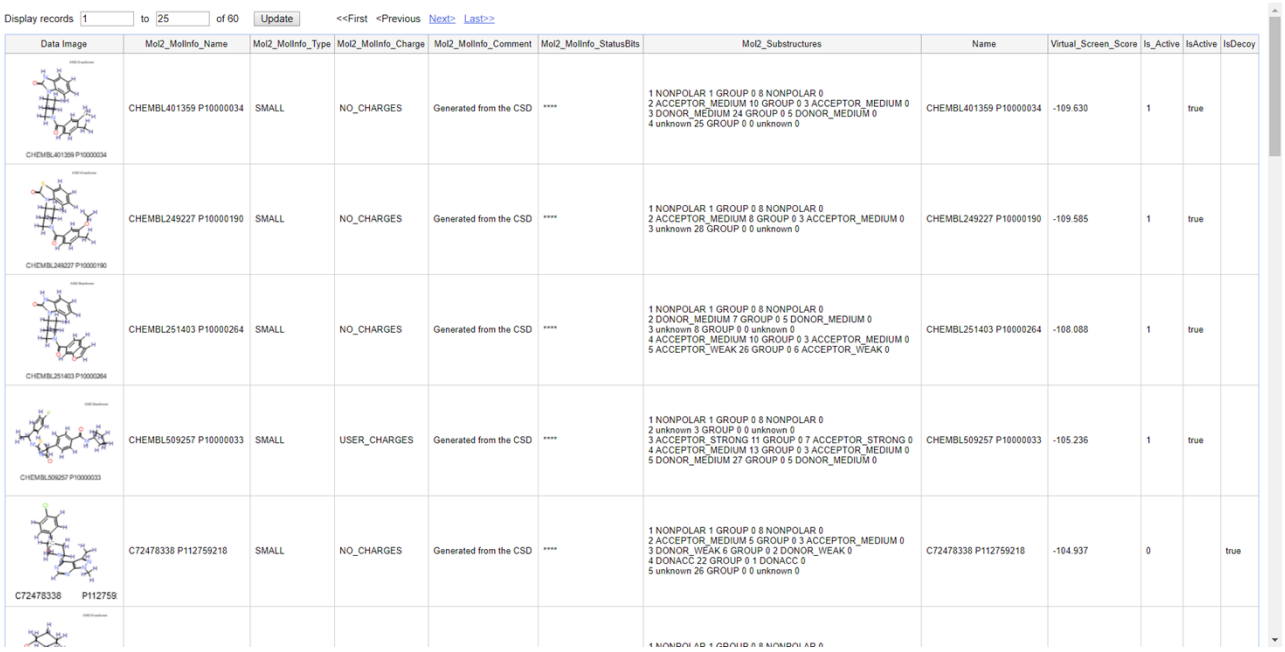
03 Screen Validation Using Tagging



This protocol takes a stream of actives (tagged), decoys (tagged), and query molecules, and performs a virtual screening validation. The output has a new property of "*Is_Active"* (1 or 0 depending upon whether the data was considered active), "*Virtual_Screen_Score"* (the score for the model).

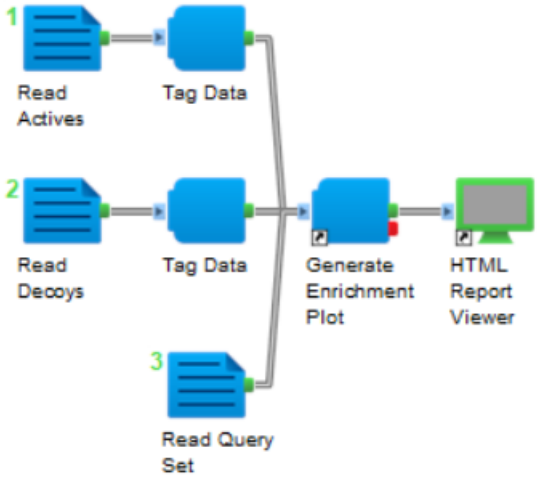


The data are then reported in an HTML page.

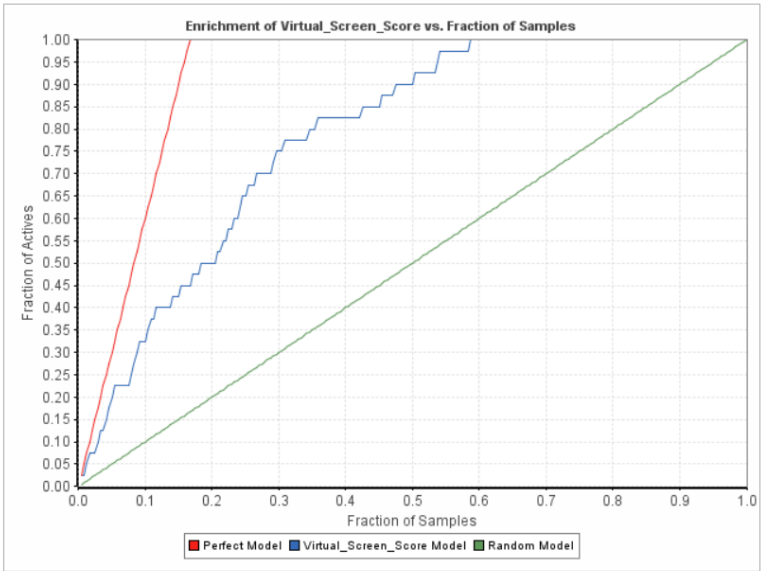


This data can be plotted in ROC or enrichment plots to reveal the selectivity of the model. There are separate components which offer those capabilities.

04 Generate Enrichment Plot Example

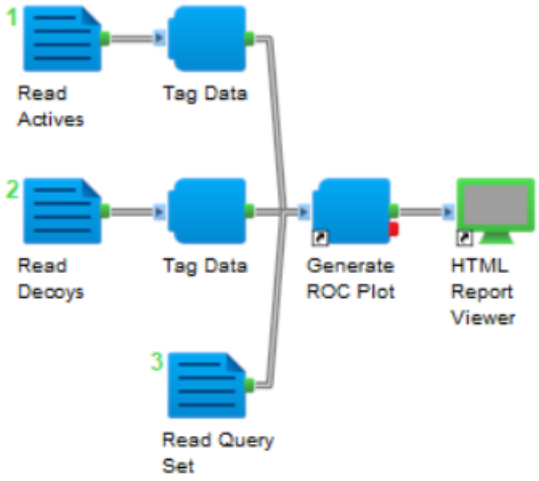


As for the *03 Screen Validation Using Tagging* protocol, this example takes a stream of actives (tagged), decoys (tagged), and query molecules, and performs a virtual screening validation however, it uses the data produced to generate an enrichment plot that is useful to help in the understanding of the selectivity of the model. The enrichments at 0.5%, 1%, 2%, and 5% is calculated and an HTML report is returned.

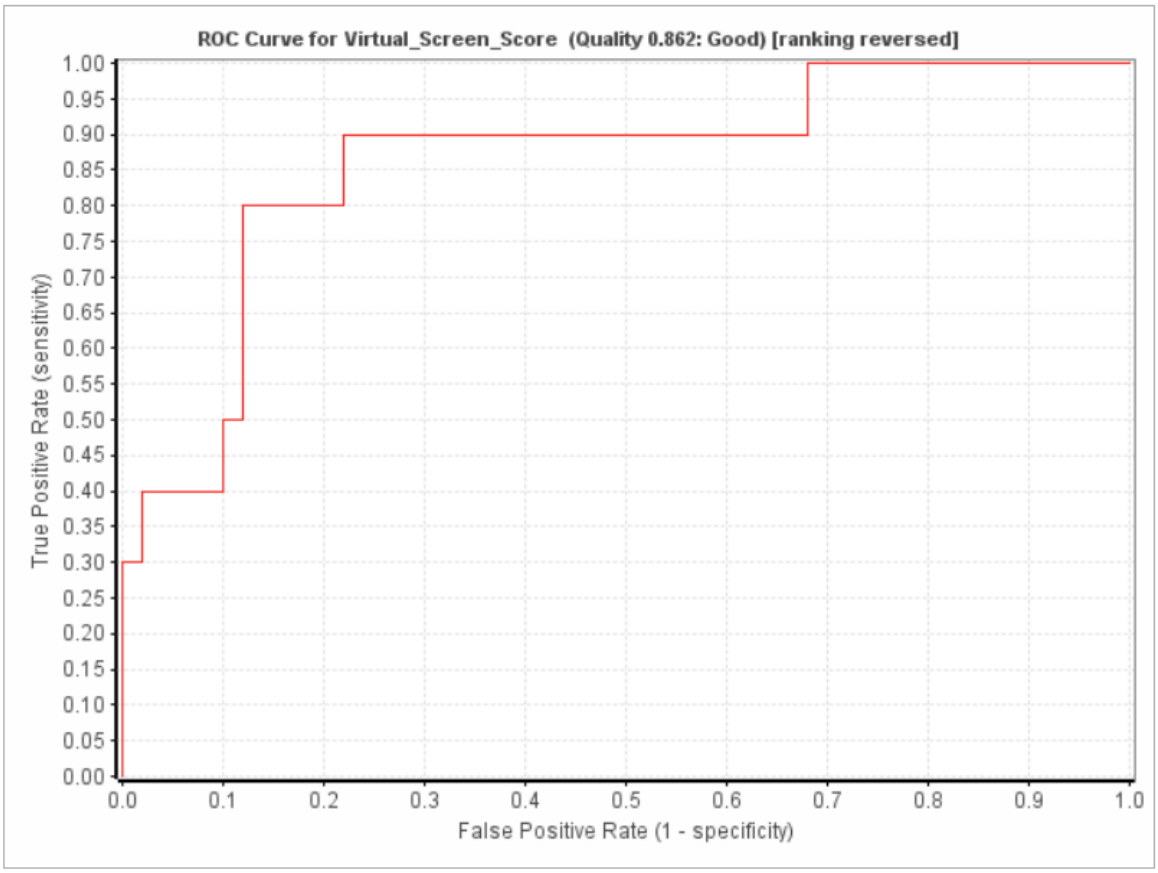


EF0.5%	EF1%	EF2%	EF5%
5.000	7.500	6.250	5.910

05 Generate ROC Example



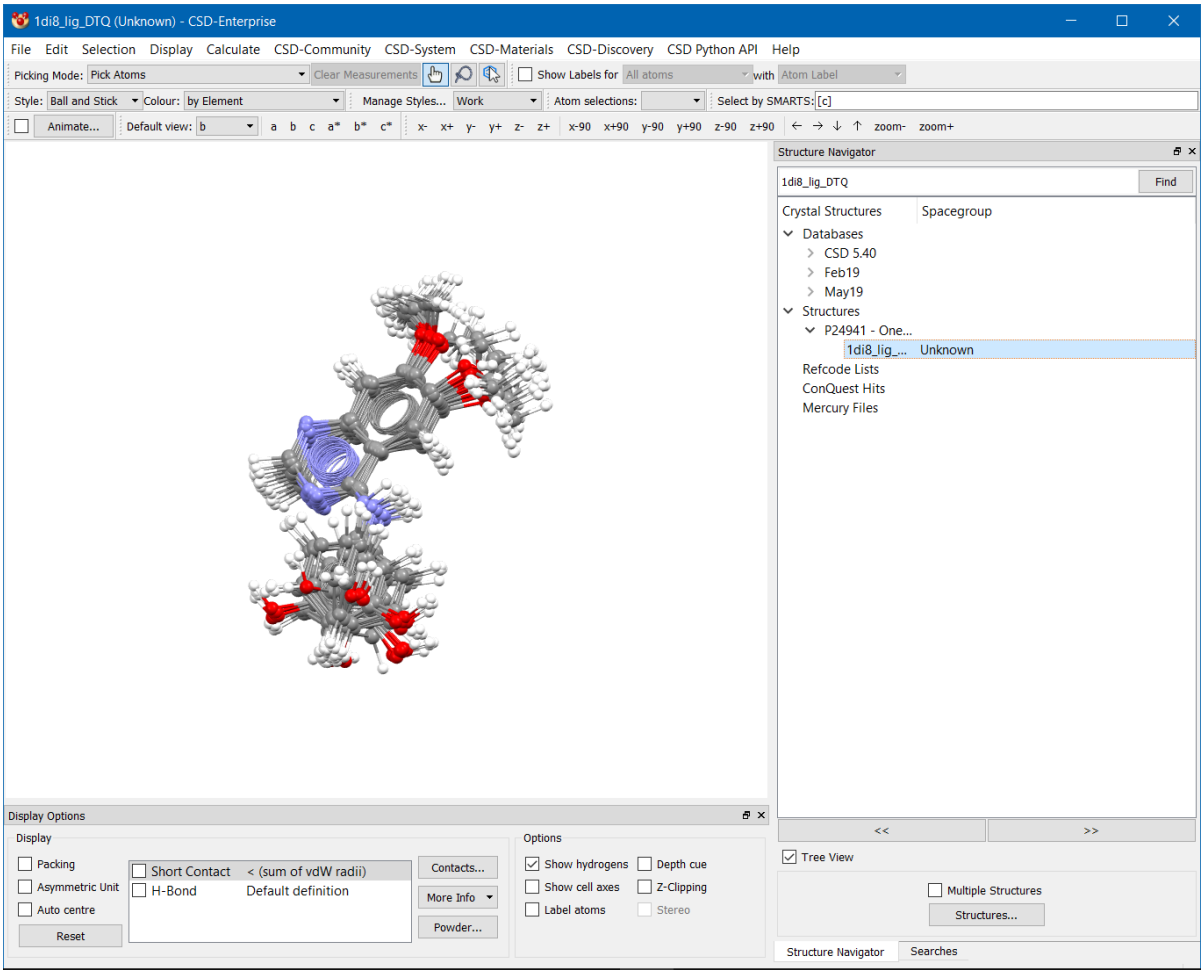
This protocol takes a stream of active molecule (tagged), decoys (tagged), and query molecules, performs a virtual screening validation and, using the data produced, generates a ROC plot. This is useful to help in the understanding of the overall efficiency of the model to separate active ligands from inactive molecules. Note that, in this example, the **Maximum** parameter is set to ten for actives and fifty for decoys.



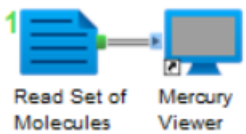
06 Generate Conformers for Molecule

This protocol takes a molecule and generates up to twenty-five conformers for that molecule, using conformer generation in the CSD Python API.

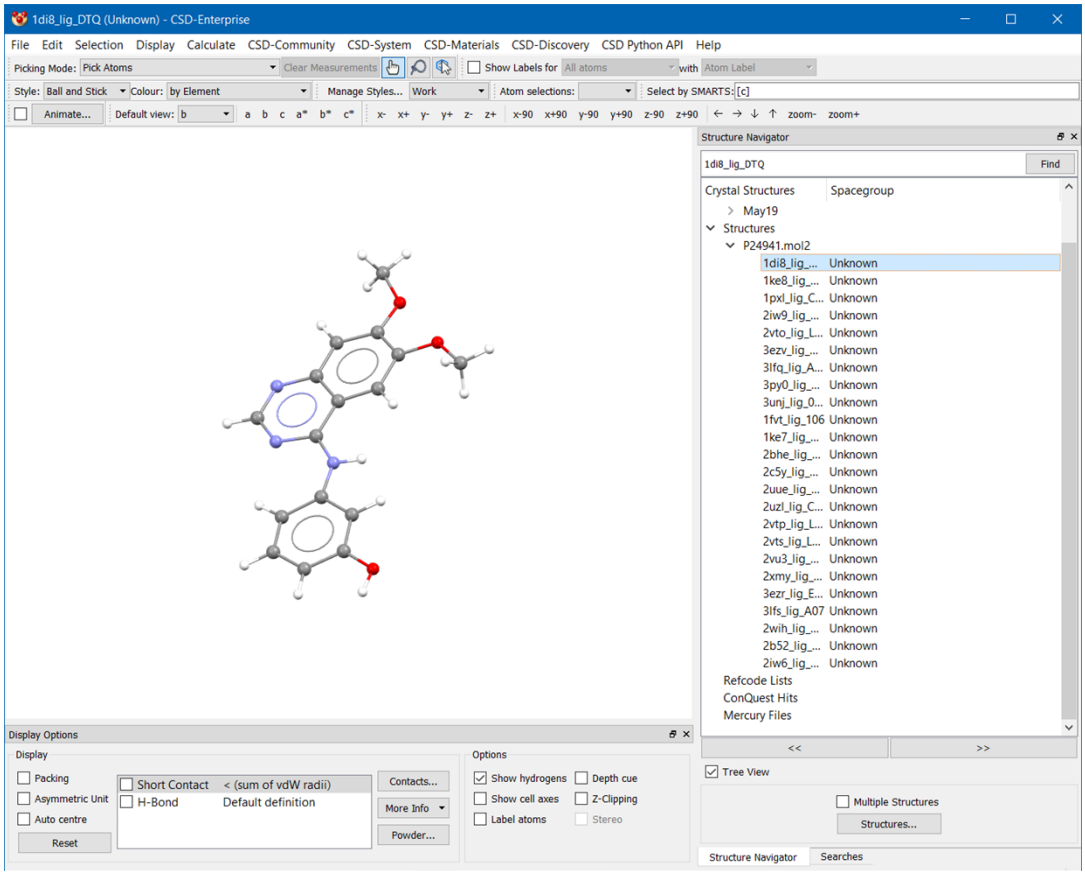
The query molecule used in this protocol is provided as part of the CSD PP Component Collection (`data\Python API Example Data\P24941sdf`). The generated conformers are superimposed on one another as specified by switching the **Superimpose** parameter in the **Conformer Options** parameters to "*True*". The overlaid results are then viewed in Mercury.



07 Mercury Viewer Example



This simple example demonstrates how molecular records can be piped into Mercury to be viewed. As the structures often contain 3D coordinates, it makes sense that they should be viewed in a way that makes sense of this. The set molecules used in this protocol is provided as part of the CSD PP Component Collection (`data\Python API Example Data\P24941sdf`).



08 Conformer Writer Example

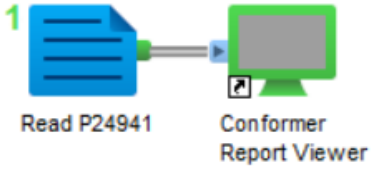
This protocol generates up to twenty-five conformers for the first three molecules provided by the incoming record file and split the molecular output. This is achieved by setting the **Spit output** parameter of the *Conformer Writer* component to "*True*". The set molecules used in this protocol is provided as part of the CSD PP Component Collection (`data\Python API Example Data\P24941sdf`)

In addition to the conformers, the summary of the conformer generation is also provided as specified in **Output Options** parameter of the *Conformer Writer* component. The conformer summary file is then displayed in an HTML report page.

Conformer Summary

Molecule name	max_log_probability	min_log_probability	n_conf_gen_clust	n_rotamers_with_no_observations	rotamers_with_no_observations
1di8_lig_DTQ	-2.437	-20.366	25	0	[]
1ke8_lig_LS4	-5.279	-24.779	25	0	[]
1pxl_lig_CK4	-7.092	-13.163	25	0	[]

09 View Conformers in Report Viewer



This protocol represents an example of how to generate a conformer generation report. In this example, the *Conformer Report Viewer* component generates up to twenty-five conformers for the first three molecules of the incoming data, then it produces an HTML report from which the results files can be downloaded. The set molecules used in this protocol is provided as part of the CSD PP Component Collection (`data\Python API Example Data\P24941sdf`). The result files included in the report are specified in the **Output What** of the *Conformer Report Viewer* component. In the example here, the molecules and the summary of the conformer generation process are generated.

Conformer Generation Report

Settings

Conformer Settings

Names	Values
Max Number of Conformers	25
Number of Threads	1
Maximum Unusual Torsions	2
Superimpose	False

Summary

Summary Table

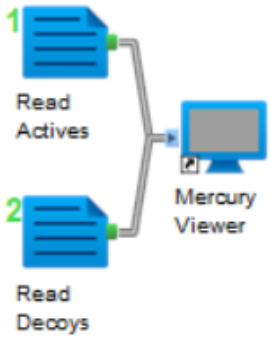
Molecule name	max_log_probability	min_log_probability	n_conf_gen_clust	n_rotamers_with_no_observations	rotamers_with_no_observations
1di8_lig_DTQ	-2.437	-20.366	25	0	[]
1ke8_lig_LS4	-5.279	-24.779	25	0	[]
1pxl_lig_CK4	-7.092	-13.163	25	0	[]

Downloads

The following file(s) were created.

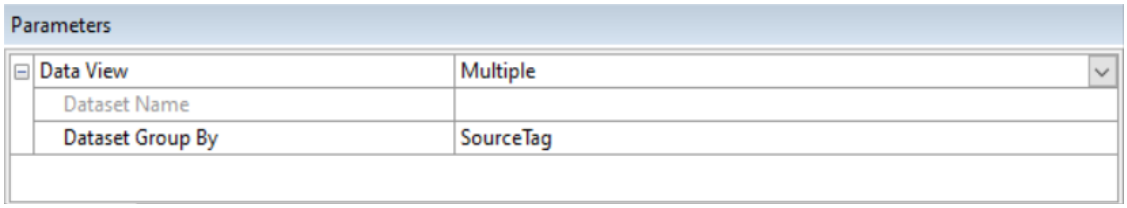
- summary.txt
- conformers.sdf

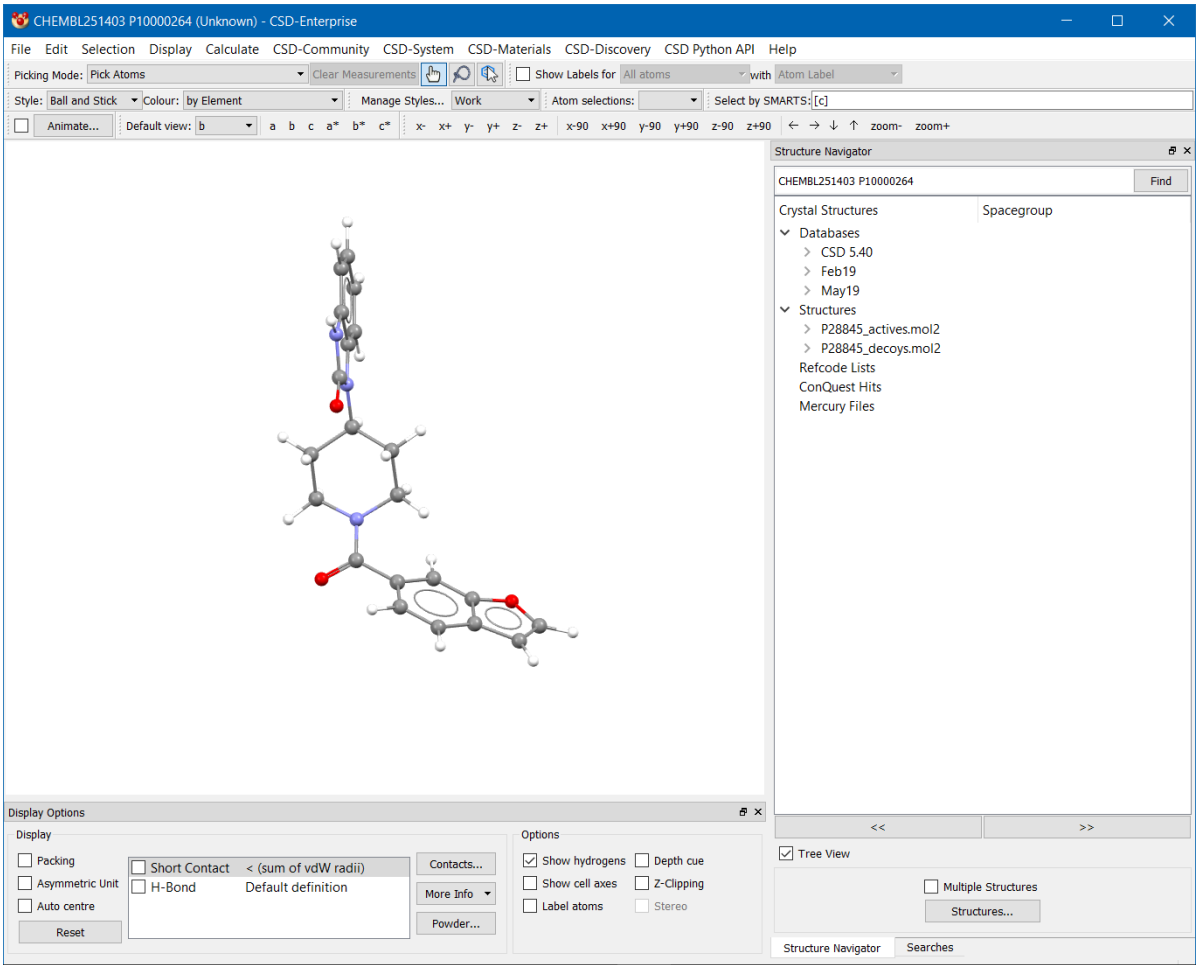
10 Mercury Viewer Example with Grouping



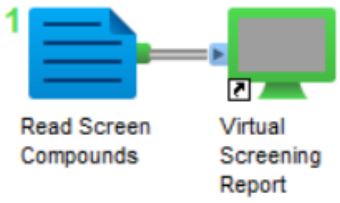
In this example, the Mercury viewer is used to visualise the data from each source specified in *Read Actives* and *Read Decoys* components but in separate groups in the Mercury Structure Navigator.

To achieve this, the **Data View** parameter in the *Mercury Viewer* component is set to "*Multiple*". This enables the **Dataset Group By** parameter. This parameter requires the name of a property by which data will be separated. In this example the **Dataset Group by** parameter is set to "*SourceTag"* that is defined as "*Filename*" in the *Read Actives* and *Read Decoys* components.





11 Virtual Screening Report Viewer Example



This protocol represents an example of how to generate a virtual screening report. In the example here, the *Virtual Screening Report* component takes a stream of the first ten incoming molecules provided (the file is available in `data\Python API Example Data\P28845_actives.sdf`), and screens against a query set identified by file (provided in `data\Python API Example Data\P28845.sdf`), then it produces an HTML report from which the *Results Files* together with the *Screening File,* the *Query File* and the *Ranked Scores File* can be downloaded.

Virtual Screening Report

Settings

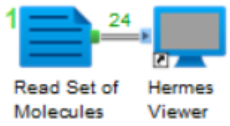
Conformer Settings	
Names	Values
Max Number of Conformers	25
Number of Threads	1

Downloads

The following files were created or used to generate the results.

- Screening File
- Query File
- Ranked Scores File
- Results File

12 Hermes Viewer Example – Structures



This simple example demonstrates how molecular records can be piped into Hermes to be viewed. In this example the molecules provided by the *Read Set of Molecules* component are displayed in Hermes.