

Working with scripts

Required functionality and modules: Discovery Studio Client.

Required data files: 1TPO.pdb.

Time: 15 minutes.

Introduction

Although Discovery Studio provides a rich set of functionality and viewers that assist in the process of performing computational analysis of data, it is often desirable to be able extend these capabilities. For example, you may need to create an analysis tool that calculates some property using a specific algorithm. You may also want to chain together a set commonly used tasks so that they can be run as a single operation.

Discovery Studio uses scripting as one way of providing this extensibility. Scripts are sets of instructions written in a programming language. Discovery Studio uses Perl as its scripting language because it is a simple and widely available programming language that is commonly used throughout the research community.

Discovery Studio provides programmatic access to much of its functionality through an Application Programming Interface (API). This API allows you to write scripts that can provide powerful functionality such as:

- Loading molecules and sequences from different file formats
- Set the display styles of objects in different Windows
- Create and execute multi-step tasks in a single operation
- Perform complex analyses of data

For additional information about Scripting and to access the API, see [Working with scripts](#).

In addition to the API, you can easily create a new command that runs a script and expose it in the application as a menu item, toolbar button, or tool on a tool panel. This makes it possible to extend Discovery Studio to suit specific needs. Scripts can also be run outside of the client, at the command line or by Pipeline Pilot components.

In this tutorial you will learn how to write and execute scripts in Discovery Studio.

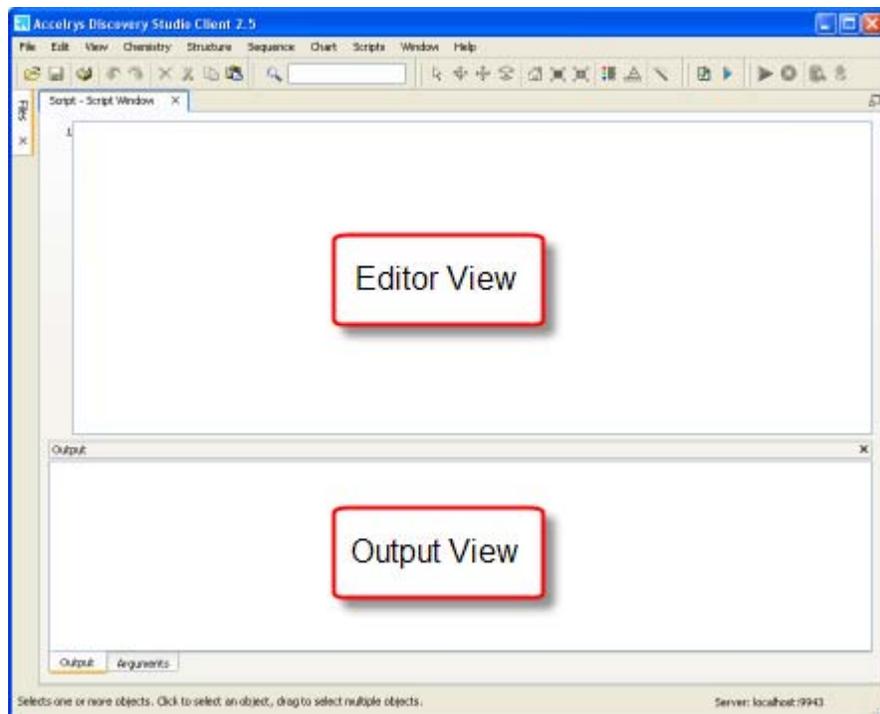
- [Create and run a simple script](#)
- [Modify a script to process data in the Molecule Window](#)
- [Create a button to run a script](#)

Create and run a simple script

Discovery Studio provides a special window, the Script Window, for creating and running scripts. The Script Window contains a number of features that assist in the development of valid perl scripts, and simplify debugging and running them.

From the menu bar, choose **File | New | Script Window**.

This opens an empty script window.



The Script Window contains two primary views:

- Editor View: This is the region of the window into which you type your script. If you open a script it also appears in the Editor View.
- Output View: As you perform operations in the Script Window, the Output View displays any information generated by those operations. For example, if you run a script that writes output data, that data appears in the Output View as the script executes. Similarly, when you check the syntax of a script, details about errors in your script are written to the Output View.

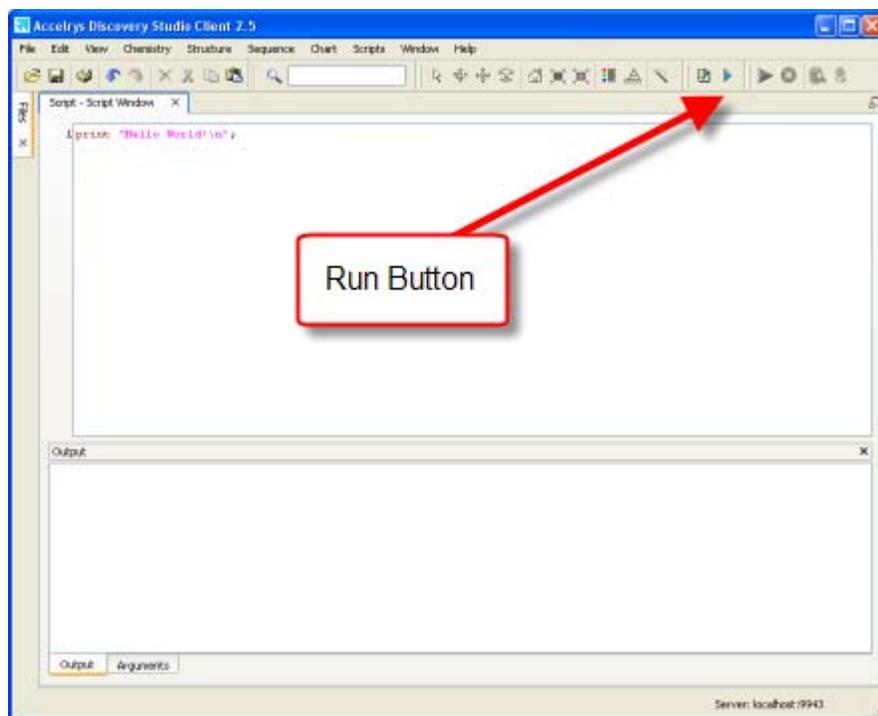
The next step is to write a simple script that prints a test string "Hello World!"

In the Editor View type the following line:

```
print "Hello World!\n";
```

On the Script toolbar, click the  **Run** button.

Notice that *Hello World!* is written in the Output View. The *Run* button launches the Perl executable, which interprets and executes the script. The executable is also referred to as the Perl interpreter.



Now write a script that utilizes the DiscoveryScript API. In this example, you will write a script that creates and displays an ethene molecule. To incorporate the DiscoveryScript API into a script, some additional set up is recommended when writing the script.

Delete the existing text in the Editor View.

Type in the following lines:

```
#!/usr/bin/perl -w
use strict;
use MdmDiscoveryScript;
```

The first line, `#!/usr/bin/perl -w` is required if the script is to be run outside of the application on Linux. It tells the Linux shell to locate the `/usr/bin/perl` program. The file is then passed to `/usr/bin/perl` for execution. The `-w` flag at the end of the line tells the Perl executable to report suspicious statements in the code that may prevent the script executing successfully.

The second line, `use strict;`, is a Perl pragma command. This is a special type of command that tells the Perl interpreter to handle the code execution in a particular way. It is not necessary to go into the details of this here, but the `use strict;` pragma tells the Perl interpreter to be strict about the way it interprets the code. In particular, ambiguous statements that could normally be handled by the interpreter are not allowed so that correct execution is more likely.

The third line, `use MdmDiscoveryScript;` is another pragma that tells the interpreter that the script uses the `MdmDiscoveryScript` module. This module is the library that implements the functionality in the API. There are several modules that make up the DiscoveryScript API, including:

- `MdmDiscoveryScript`: Contains functionality for accessing the Molecular Data Model (MDM).
- `SdmDiscoveryScript`: Contains functionality for accessing the Sequence Data Model (SDM).
- `PharmacophoreDiscoveryScript`: Contains functionality for working with pharmacophore data.
- `SbdDiscoveryScript`: Contains functionality for working with binding sites and other functionality relevant to Structure Based Design (SBD).

For more information about available modules and their functionality, consult the Discovery Studio help.

Now add some code that will create a document to display the molecule. In this case, a document is a window, such as a Molecule Window, and the data it contains. Discovery Script supports several document types such as MDM and SDM documents.

Add the following line to the script in the Editor Window:

```
my $document = Mdm::Document::Create();
```

This line instructs the program to create an MDM document. If the script is being run within the application, the result of adding this line will be to create a new Molecule Window. Otherwise, this script has no observable effect when run outside of the application (e.g., in a standalone script).

The objective is to create a simple ethene molecule. The next line in the script creates the molecule object. At this point, the molecule has no atoms, bonds, or other objects, so it is effectively an empty container. There will be no update to the screen on executing this line.

Add the following line to the script in the Editor Window:

```
my $molecule = $document->CreateMolecule();
```

At this point, the basic document and molecule object have been set up and it is possible to use MdmDiscoveryScript to create and assemble the objects that make up the molecule.

Add the following lines to achieve this:

```
my $atom1 = $molecule->CreateAtom('C');
my $atom2 = $molecule->CreateAtom('C');

my $ccBond = $document->CreateBond($atom1, $atom2, Mdm::doubleBond);

$document->AddHydrogenAtoms();

$document->Clean();
```

The first two lines create variables for the two carbon atoms. The third line creates a double bond between the two atoms. The fourth line adds hydrogens to the carbon atoms. The final line invokes the Clean method, which uses a fast, Dreiding-like forcefield to improve the geometry of the selected atoms and results in an approximate 3D structure.

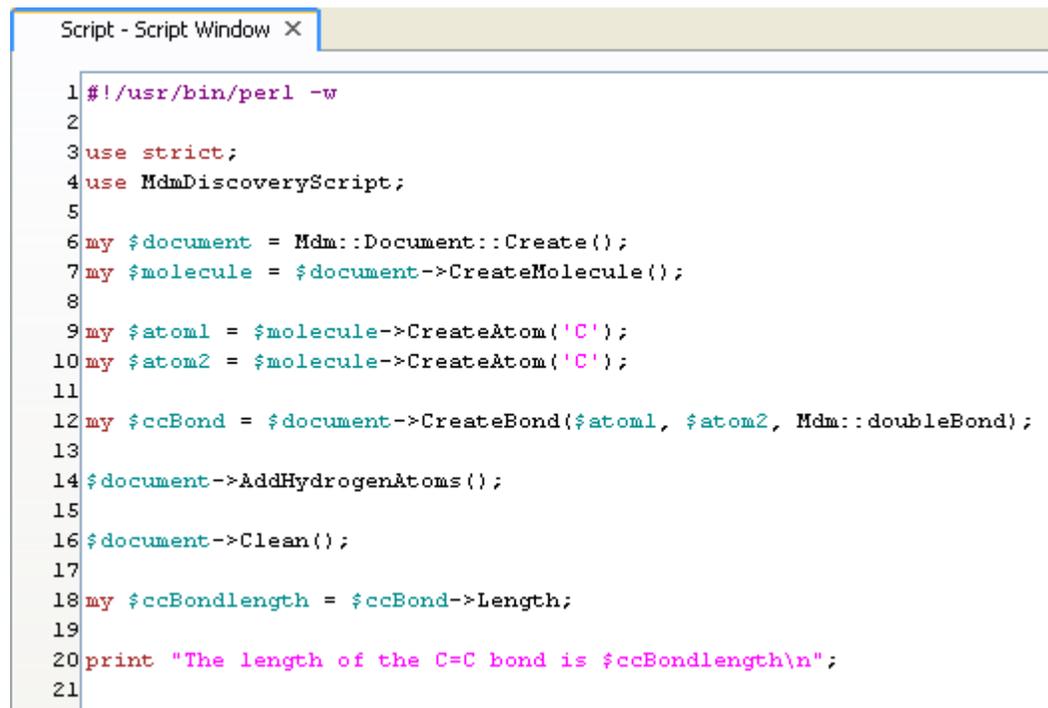
At this point, the molecule has been created and now you can add lines to report information about the bond length between the carbon atoms in the 3D structure.

Add the following two lines:

```
my $ccBondlength = $ccBond->Length;

print "The length of the C=C bond is $ccBondlength\n";
```

The Editor View should appear as follows:



```
Script - Script Window X
1#!/usr/bin/perl -w
2
3use strict;
4use MdmDiscoveryScript;
5
6my $document = Mdm::Document::Create();
7my $molecule = $document->CreateMolecule();
8
9my $atom1 = $molecule->CreateAtom('C');
10my $atom2 = $molecule->CreateAtom('C');
11
12my $ccBond = $document->CreateBond($atom1, $atom2, Mdm::doubleBond);
13
14$document->AddHydrogenAtoms();
15
16$document->Clean();
17
18my $ccBondlength = $ccBond->Length;
19
20print "The length of the C=C bond is $ccBondlength\n";
21
```

Notice that the lines are numbered. Also notice that text elements are colored differently. This is known as syntax

highlighting. You can control the numbering and highlighting by right-clicking the Script Window and choosing *Display Style...*

Before running the script, it is useful to review two important features of the Script Window: the Syntax Checker and context sensitive help.

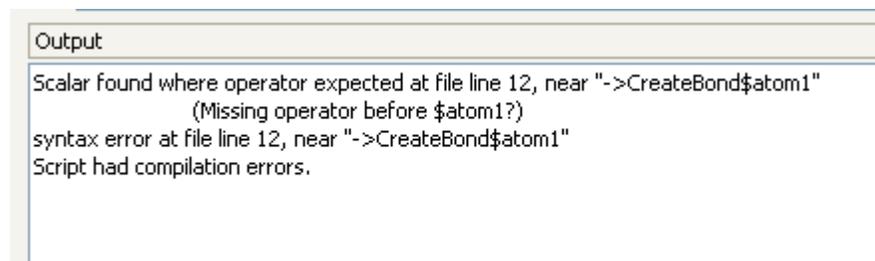
The syntax checking feature runs the Perl interpreter to identify any syntactic errors in a script. To illustrate this, edit the current script to introduce an error.

On line 12 of the script, delete the first parenthesis in the CreateBonds function call:

```
my $ccBond = $document->CreateBond$atom1, $atom2, Mdm::doubleBond);
```

Press the  **Check Syntax** button on the Script toolbar.

The syntax checker will detect the error and report it in the Output View:



Another powerful feature of the Script Window is the context help. You can select a keyword in a script and press the F1 key to display help about it. In general, help provides an explanation of the functionality and an example of its use.

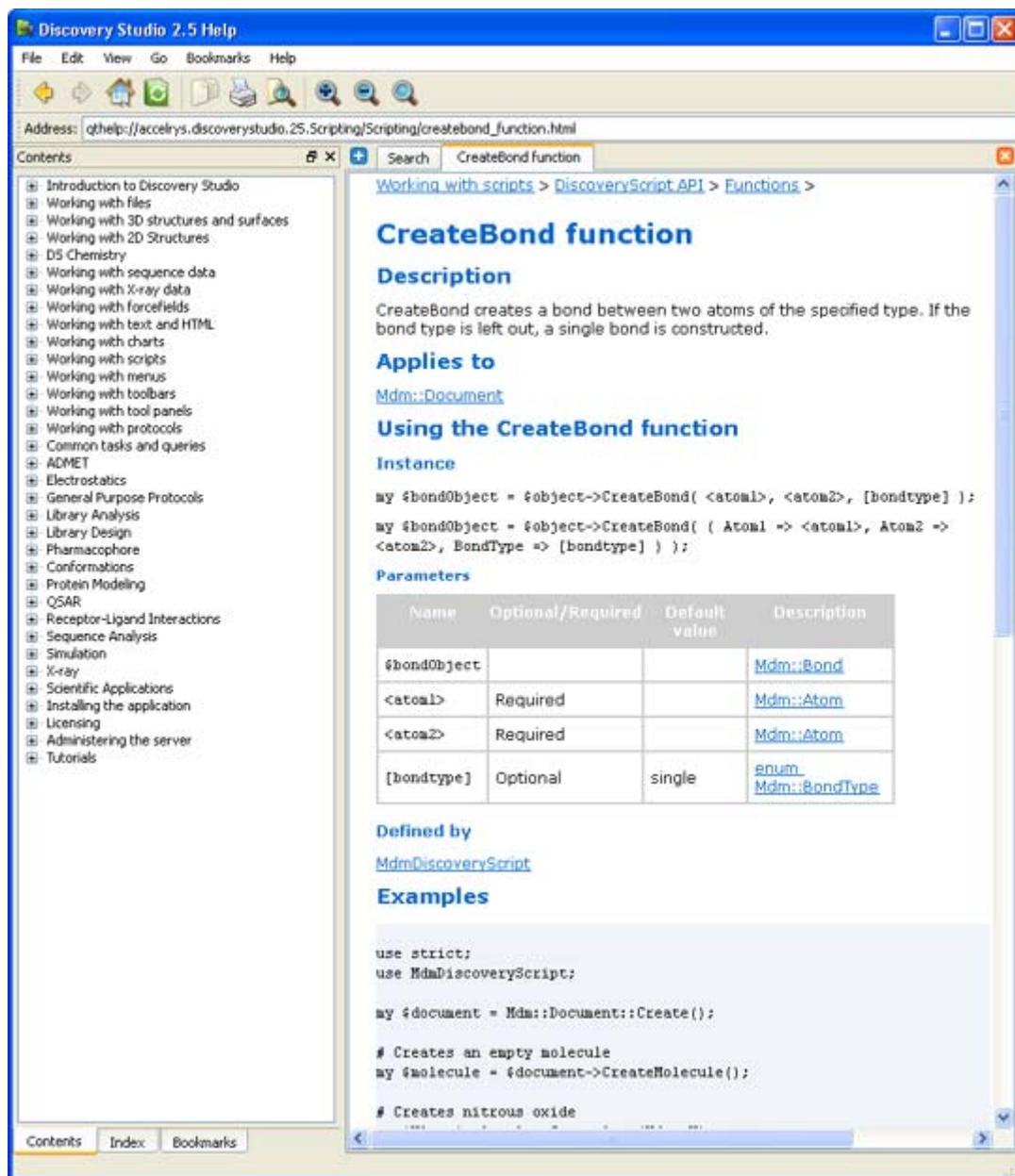
On line 12 of the script correct the error by re-inserting the first parenthesis in the CreateBonds function. To do this, press **Ctrl+Z** to undo the previous edit. This should change the code back to:

```
my $ccBond = $document->CreateBond($atom1, $atom2, Mdm::doubleBond);
```

Double click the CreateBond text to select it.

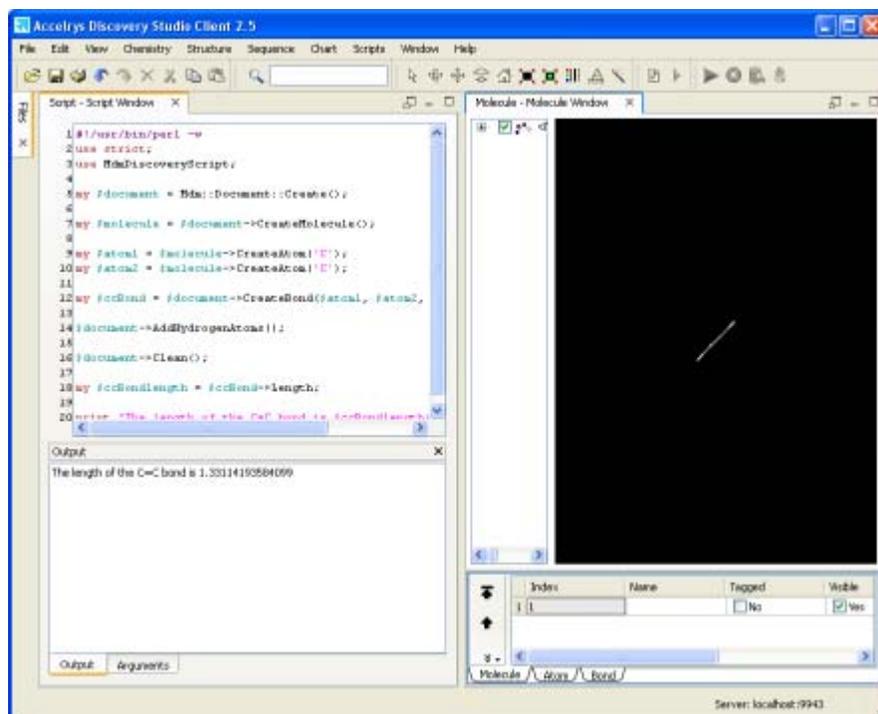
Press the **F1** key.

The Discovery Studio Help window will open and display help on the CreateBond function.



The script is ready to run.

On the toolbar, click the  **Run** button. This results in the creation of a new Molecule Window containing the Ethene molecule. Drag the tab on the Molecule window to position the Molecule Window next to the Script Window.



Inspect the molecule created in the Molecule Window to verify it is an ethene molecule. Also note that the bond length has been written in the Output Window.

At this point, you can save the script if desired.

Modify a script to process data in the Molecule Window

The first section of this tutorial demonstrated how to create a new molecule in the application using the DiscoveryScript API. However, a powerful feature of the API is the ability to reference an object in memory and modify or analyze the data. For example, you can write a script that accesses the data object corresponding to a molecule already displayed in the application and make changes to it (e.g., modify the appearance). Similarly, you can write a script that references a sequence alignment and performs some analysis.

Now you will create and run a script that modifies a protein molecule loaded in the Molecule Window. The script will update the appearance of a protein structure to prepare it for publication.

From the main menu bar choose **Windows | Close All** to close all open windows in the application workspace.

From the Files Explorer, open **Samples | Tutorials | Quick Start Tutorials | 1TPO.pdb**.

This file is a raw pdb file. The default display shows only the atoms for the structure. You will now write a script to display the molecule using a Schematic Display for the protein chains.

From the menu bar select **File | New | Script Window**.

Drag the tab on the Script window to position the Script Window next to the Molecule Window.

Copy and paste the following script into the Script Window:

```
#!/usr/bin/perl -w

use strict;
use MdmDiscoveryScript;

# Create a document from the last active Molecule Window
my $document = DiscoveryScript->LastActiveDocument(MdmModelType);

# Ensure all view updates turned off so action seems atomic
$document->EnableUpdateViews(False);

# Define variables of the display styles that will be applied
my $atomDisplayStyle = Mdm::styleAtomNone;
```

```

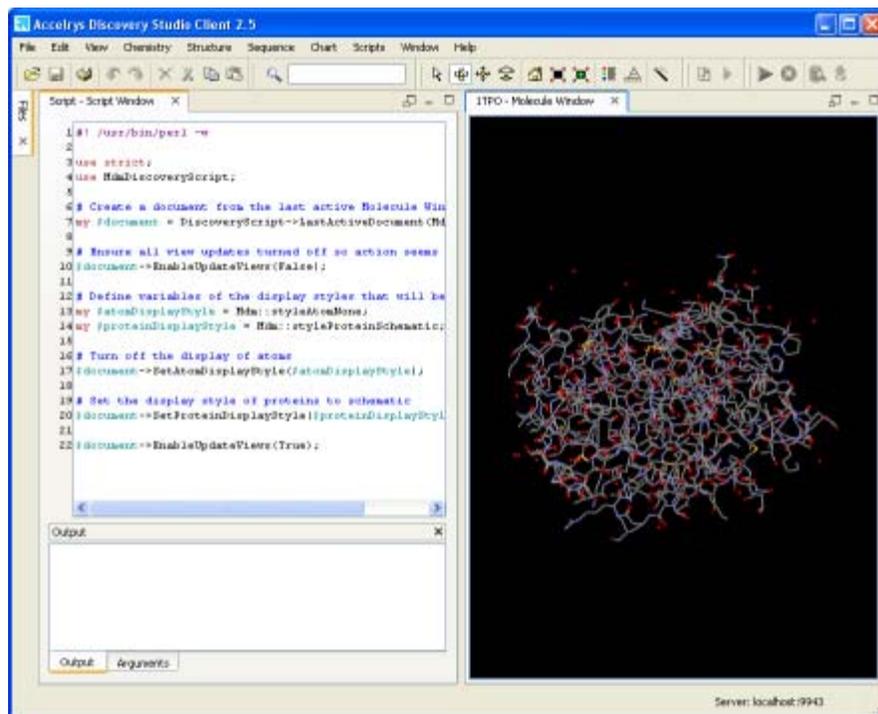
my $proteinDisplayStyle = Mdm::styleProteinSchematic;

# Turn off the display of atoms
$document->SetAtomDisplayStyle($atomDisplayStyle);

# Set the display style of proteins to schematic
$document->SetProteinDisplayStyle($proteinDisplayStyle);

$document->EnableUpdateViews(True);

```



This code snippet emphasizes a number of important features of scripting. Review line 7:
`my $document = DiscoveryScript->LastActiveDocument(MdmModelType);`

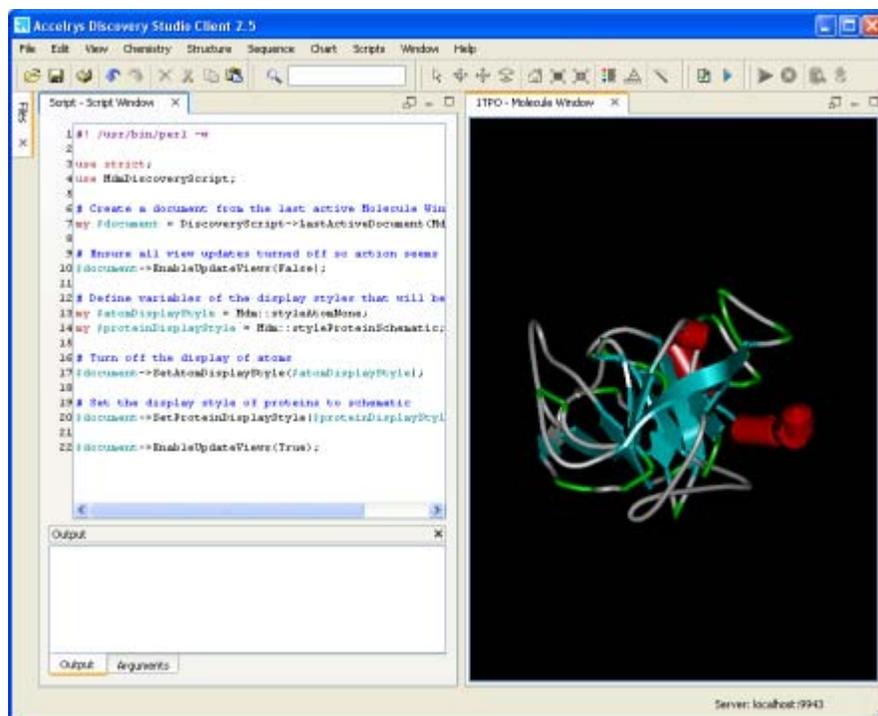
The function `LastActiveDocument()` is a powerful command that cycles through the current open windows and finds the last open document matching the model type, in this case `MdmModelType`. The function then sets the `$document` variable to that Mdm object. At this point, the data object can be directly manipulated by the script by using the variable.

Now review line 9:

```
$document->EnableUpdateViews(False);
```

The `EnableUpdateViews()` function turns on or off any updates to the display. This is useful when a number of operations are to be performed that result in updates to the display, but the script is intended to update the display in a single action (i.e., without any flickering and for improved performance). In this code snippet, the display updates are disabled until the display refresh is turned back on in line 22.

The main work of the script is done in lines 17 and 20. Here `SetAtomDisplayStyle` and `SetProteinDisplayStyle` apply the specified styles to all molecules in the document. If desired, this section of the script could be enhanced to apply other display effects, but only these two operations are shown in this example.



Now save the script for use in the next section of this tutorial.

From the menu bar, choose **File | Save As...**

Save the script with the name **ApplyProteinSchematic.pl**.

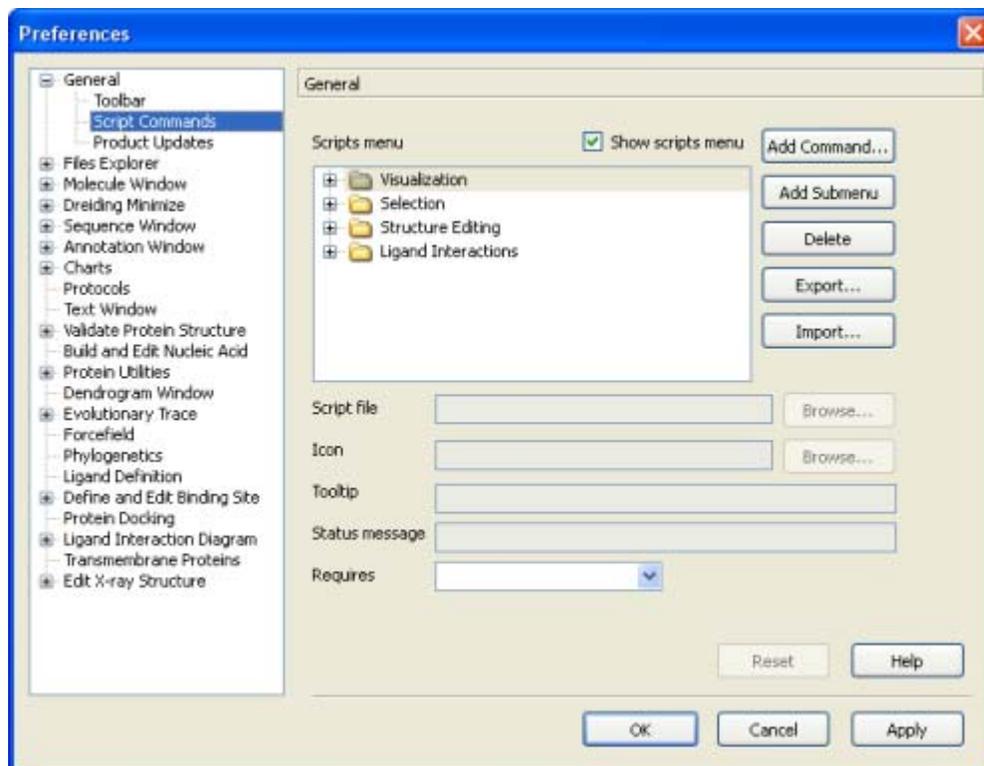
Create a button to run a script

Discovery Studio provides a powerful interface for creating tools from scripts that can be launched from menus, toolbars, and toolpanels. You can create these tools using the *Script Commands* preference page. This page enables you to select a script that can run inside the application and assign attributes to it such a tool tip, status message, and icon.

In this section, you will create a toolbar button that launches the script you just created.

From the menu bar, choose **Edit | Preferences** to open the *Preferences* dialog.

Open **General | Script Commands**.



Click **Add Command** to open the *Choose Script* dialog.

Navigate to the location in which the `ApplyProteinSchematic.pl` was saved and select it.

Click **Open**.

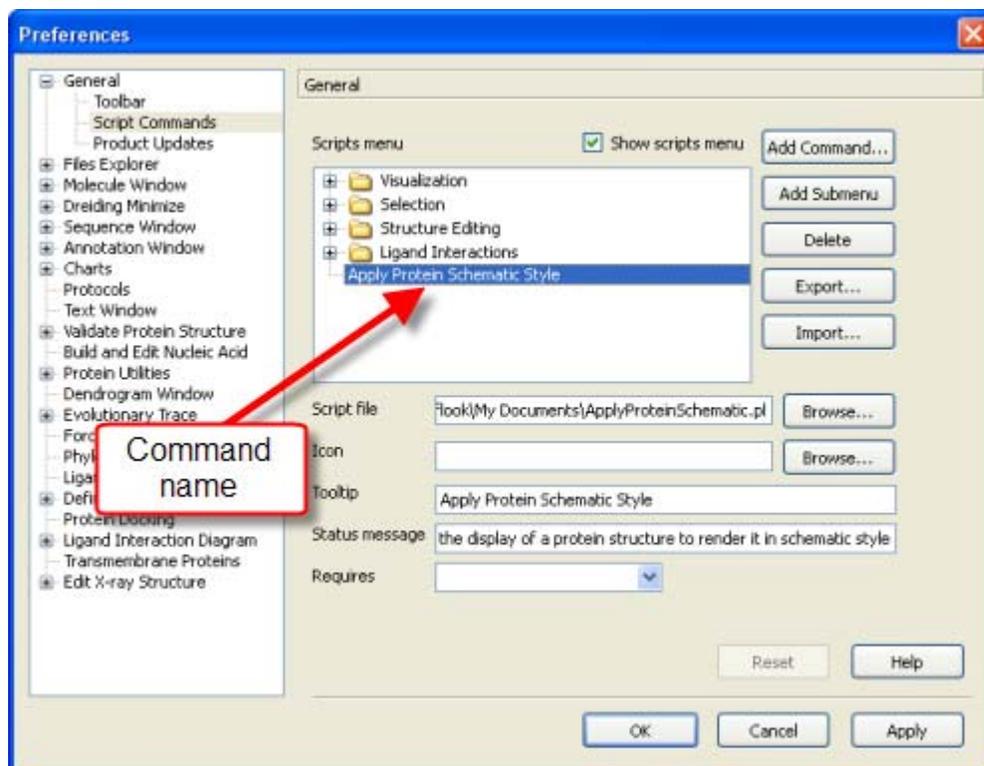
This specifies the `ApplyProteinSchematic.pl` as the source script for the command.

In the **Tooltip** input field enter: "Apply Protein Schematic Style".

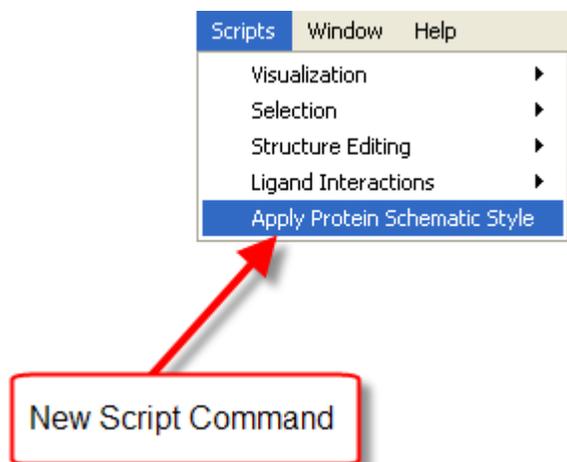
In the **Status Message** input field enter "Updates the display of a protein structure to render it in schematic style".

In the **Scripts** menu tree view, double-click new script command and type its name: "Apply Protein Schematic Style".

Click **OK**.



By default, when a new script command is created it appears in the Scripts menu. When the application is installed a set of toolbar folders are configured as shown in the screenshot below. The new script command can be dragged into one of the existing toolbar folders. Alternatively, a new toolbar folder can be created and the script command moved there.

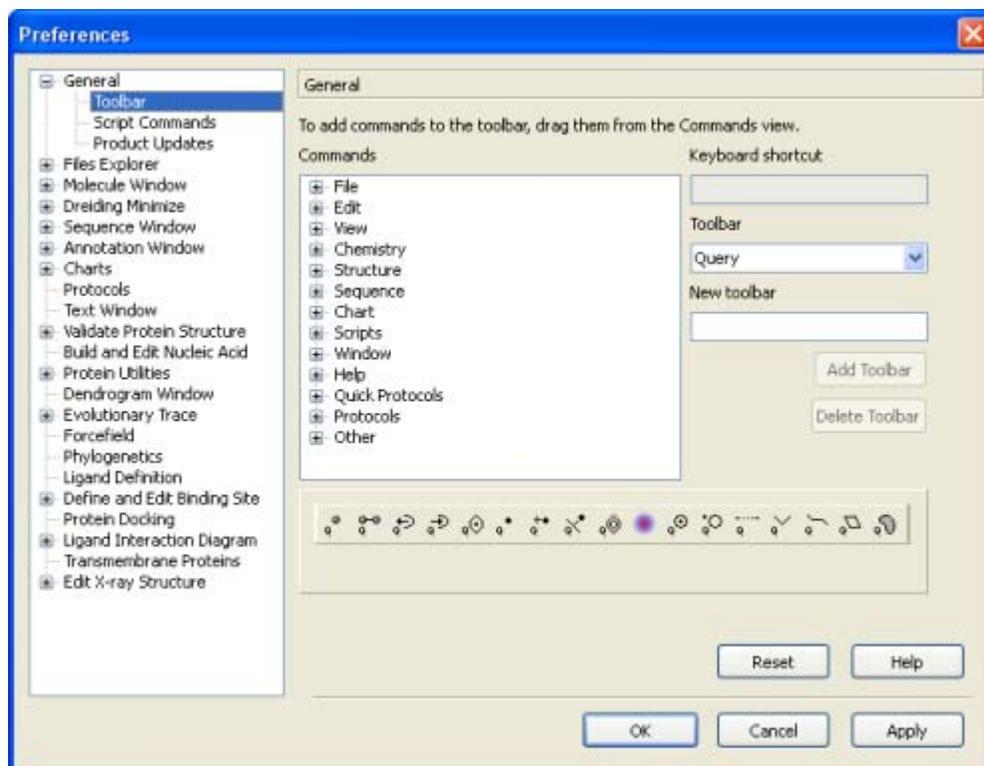


Notice that the new script is now available from the menu.

You will now create a toolbar button to launch the script command. The Toolbar preferences page will be used to create a new toolbar and button.

Choose **Edit | Preferences** from the menu bar.

Open **General | Toolbar**.

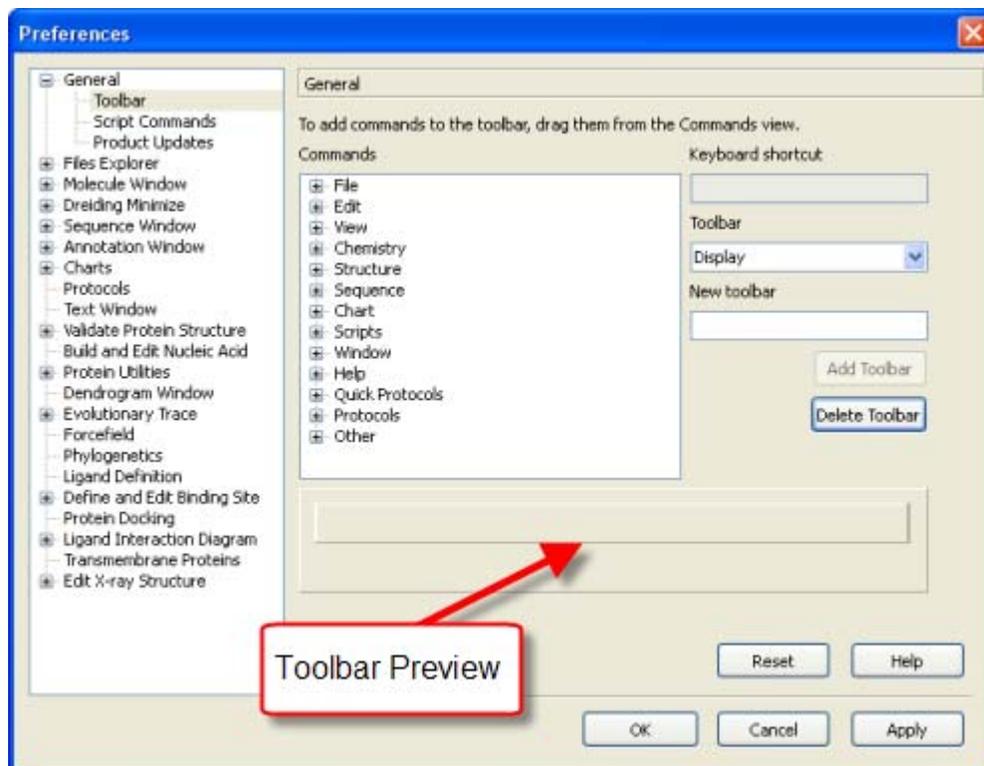


The *Toolbar* preference subpage enables the addition of new tools to any of the preset toolbars. In addition, you can use it to create new toolbars. To expose the new *Apply Protein Schematic Style* tool, you will first create a new toolbar.

In the **New toolbar** field, type "Display".

Click **Add Toolbar**.

This creates the new *Display* toolbar. The new toolbar is now selected in the *Toolbar* list. At the bottom of the dialog a blank *Toolbar Preview* is displayed.

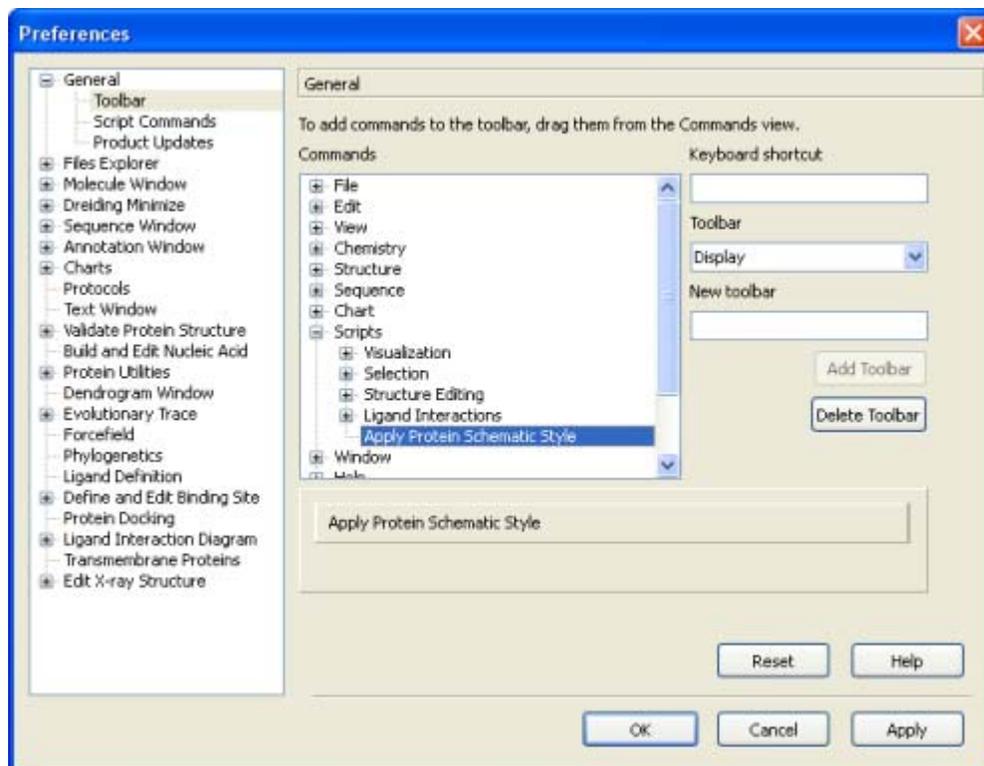


In the **Commands** hierarchy, expand **Scripts**.

Scroll down and locate the **Apply Protein Schematic Style**.

Click and drag it to *Toolbar Preview*.

Click **OK**.



A new toolbar is now displayed in the application window. This toolbar has a single command, *Apply Protein Schematic Style*.

Reopen **1TPO.pdb**

Click the new **Apply Protein Schematic Style** button to run the script.

