Skyline Interactive Tool Support

Skyline supports external tools that extend Skyline functionality without directly integrating into the large and complex Skyline source code base. External Tools provide support for:

- prototyping new functionality in a simpler code environment,
- creating a tool that might only be useful for a subset of the Skyline user community,
- incorporating proprietary code that can't be added to the Skyline open source distribution model,
- developing extensions to Skyline using a language other than C#.

Without the interactive tool support advances described here, the main disadvantage for External Tools is that they are fairly restricted in what they can do. Without interactive support tools are "run once" utilities: they analyze data transmitted from Skyline through a fixed report, and produce results, without any further opportunity for interaction between a tool and Skyline.

This document describes how to create "interactive" tools which can react to changing documents and selections in Skyline, change the selection in the Targets view or the active replicate, get data using dynamically created reports, and even access chromatogram data. With interactive tools, the distinction between functionality provided by the tool and built-in Skyline code begins to blur. As support for interactive tools continues to develop, new kinds of cooperative processing will become possible between Skyline and its External Tools.

The remainder of this document assumes you are familiar with Skyline External Tools of the non-interactive variety, as described in the External Tools documentation:

https://skyline.gs.washington.edu/labkey/docs_external_tools.url

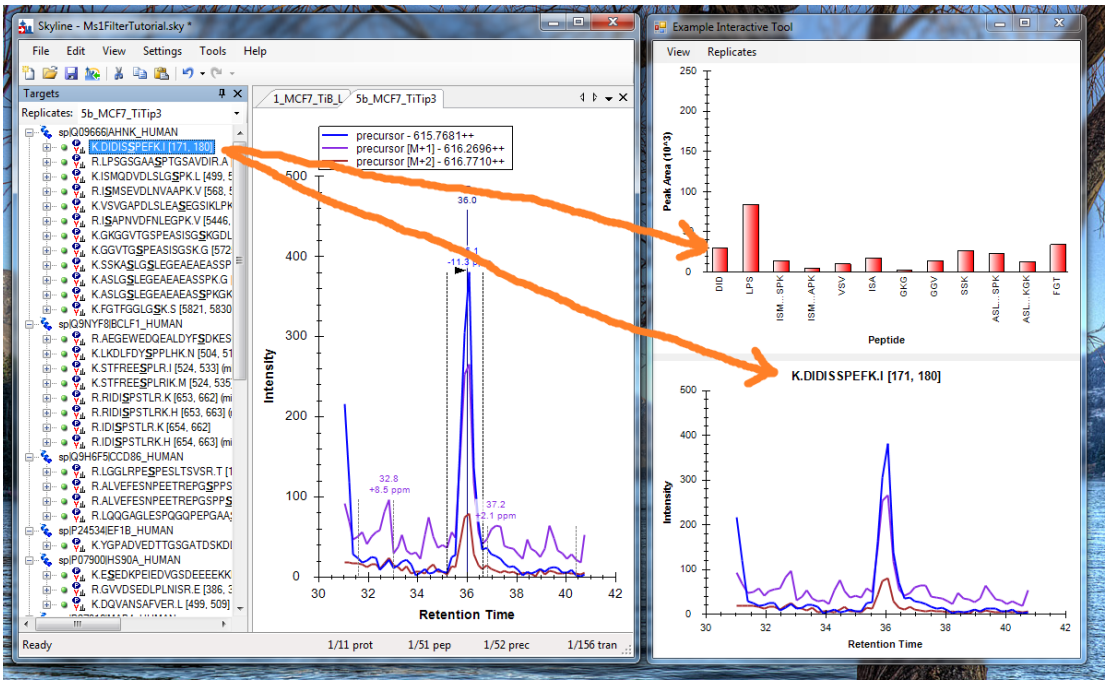
Example Interactive Tool

Many of the code features described in this document are demonstrated in an example project called ExampleInteractiveTool.sln, which can be found in this folder in the Skyline source code tree: `pwiz\pwiz_tools\Skyline\Executables\Tools\ExampleInteractiveTool`.

To demonstrate interactive tool features, Example Interactive Tool does the following things:

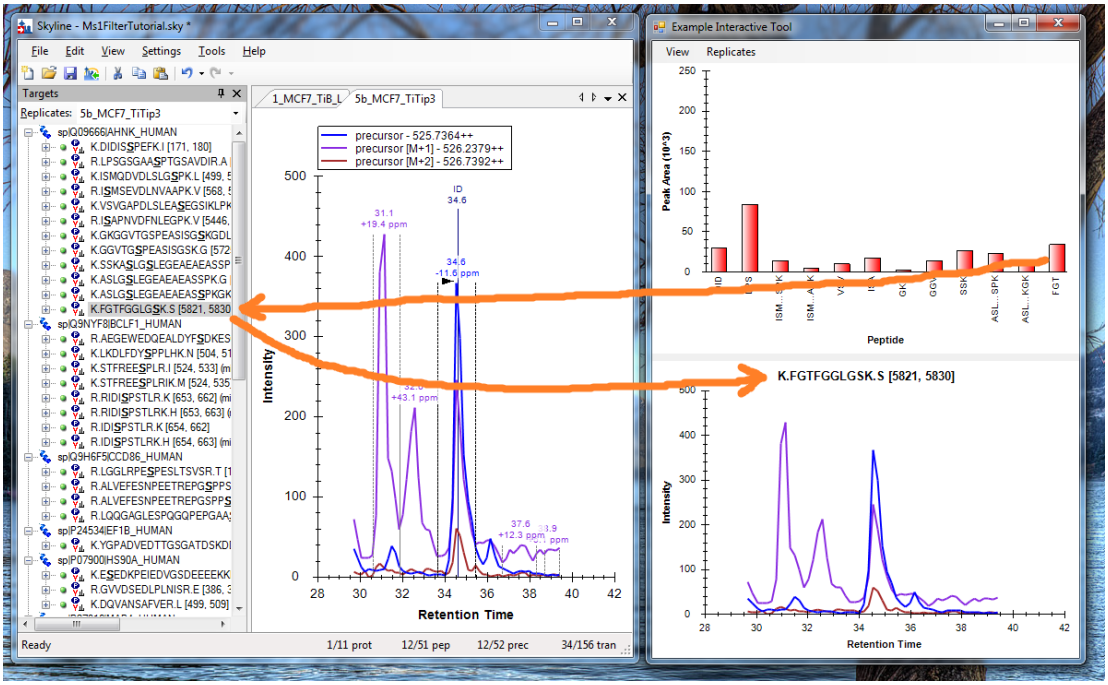
1. Presents a bar chart showing the peak areas of each peptide in a given replicate (or summed over all replicates).
2. Allows users to click on any bar to select the corresponding peptide in Skyline.
3. Shows a chromatogram of the current Skyline selection.

Here's what it looks like in action:



The peptide selected in Skyline (K.DIDISSPEFK.I) has a corresponding peak area in the bar chart shown in the tool (labeled with the unique abbreviation “DID”) and a chromatogram that looks similar to the one displayed in Skyline.

If the user clicks on one of the peak area bars (such as “FGT” on the right side of the chart), the tool tells Skyline to select the corresponding peptide. After Skyline updates the selection, it sends a message to the tool, notifying it that the selection has changed. The tool asks Skyline what the new selection is, and then requests the corresponding chromatogram, which it displays:

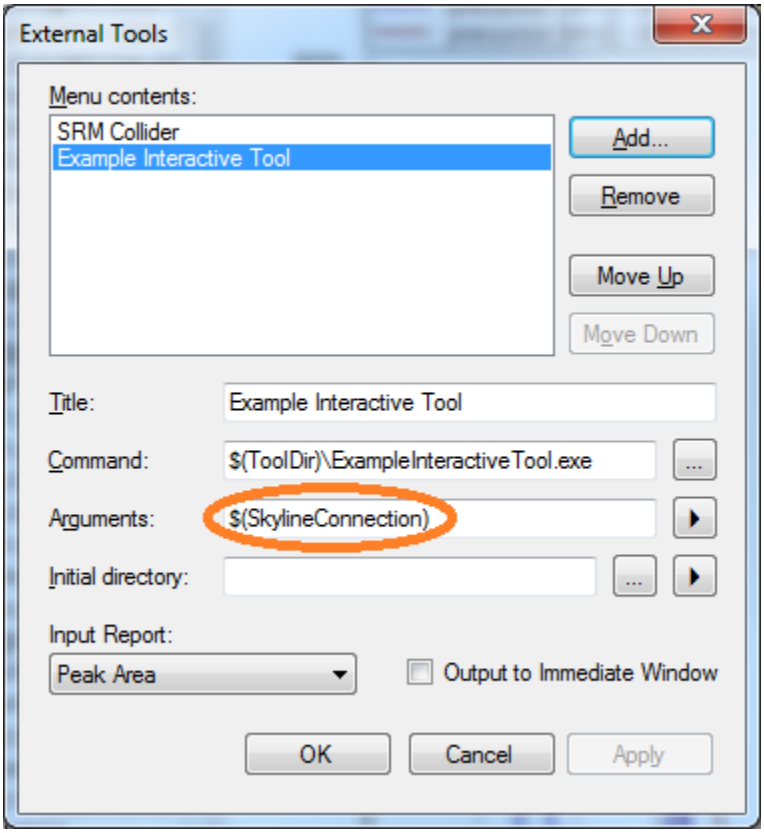


Creating an interactive tool

There are a few things you need to do enable interactive features in an external tool:

1. Include a reference to SkylineTool.dll in your project. The DLL can usually be found in the platform and configuration folders under `pwiz\pwiz_tools\Skyline\SkylineTool\bin`. Note that Example Interactive Tool also references Zedgraph.dll in order to display graphs similar to the ones in Skyline.

2. Add the tool macro **$(SkylineConnection)** to your tool's argument list:



3. Use the value of the SkylineConnection argument to construct a SkylineToolClient object:

```
// Create tool client and register for events.
if (args.Length > 0)
{
    _toolClient = new SkylineToolClient(args[0], "Example Interactive Tool");
    _toolClient.DocumentChanged += OnDocumentChanged;
    _toolClient.SelectionChanged += OnSelectionChanged;
}
```

SkylineToolClient

SkylineToolClient is a class implemented in SkylineTool.dll that provides a communication conduit between a tool and the instance of Skyline that started the tool. The **SkylineConnection** argument is the key to this conduit. Using the same key, any number of tools can communicate with a particular instance of Skyline. In theory, you could run multiple instances of Skyline, and each could communicate with multiple tools simultaneously. None of the messages would get confused. The main drawback of this design is that you can't start a tool independently of Skyline and have it automatically connect to Skyline.

SkylineToolClient keeps the connection to Skyline open until it is Disposed or Skyline exits. In order to keep things working smoothly, you should call the Dispose method of each instance of SkylineToolClient you create before your tool shuts down. If you do not (or if your tool crashes before it calls Dispose), Skyline must use heuristics to determine which tools are no longer responding. This can slow Skyline down for a while until it determines that sending further messages to a non-responsive tool is fruitless.

The methods provided by SkylineToolClient fall into the following categories:

1. Get a report from Skyline.
2. Get and set the "document location" (the current selected element) in Skyline.
3. Get the name of the current replicate.
4. Get selected chromatograms.
5. Get the path of the current Skyline document.
6. Get Skyline version information.
7. Register for document and selection events.

Getting reports

There are two methods for getting a report from Skyline:

- `IReport GetReport(string reportName)`
- `IReport GetReportFromDefinition(string reportDefinition)`

GetReport returns a report based on a pre-defined report contained in the tool as a .skyr file in the tool-inf folder. This is an easy way to implement a tool that is based on a few reports with fixed formats.

GetReportFromDefinition allows your tool to create report definitions dynamically. You can create strings from embedded resources, string format statements, or hard-coded string constants within the tool. The resulting definitions should be similar to what you would put in a .skyr file.

Example Interactive Tool uses both methods to generate the same report:

```
// Retrieve the current report.
IReport report = _toolClient.GetReport("Peak Area");

// Get the same report, more dynamically.
var reportStream = typeof(MainForm).Assembly.GetManifestResourceStream(
    "ExampleInteractiveTool.tool_inf.ExampleTool_report.skyr");
var reader = new StreamReader(reportStream);
IReport report2 = _toolClient.GetReportFromDefinition(reader.ReadToEnd());
```

The IReport object returned by these methods gives you access to column names and cell values in a few different ways:

```
public interface IReport
{
    string[] ColumnNames { get; }
    string[][] Cells { get; }
    double?[][] CellValues { get; }
    string Cell(int row, string column);
    double? CellValue(int row, string column);
}
```

The ColumnNames property returns an array of the column names in the report. Cells returns a two-dimensional array containing cell values as strings. CellValues returns the same two-dimensional array, but with numeric values already parsed as double values (or null values for cells containing non-numeric information). The Cell and CellValue methods return individual cell values accessed by row index and column name.

Skyline reports can contain all sorts of interesting information, but it's especially useful to include columns that return document location information that you can use to change selection in the Skyline Targets view. For example, the report requested by ExampleInteractiveTool contains two document location columns:

- `Precursor.Peptide.DocumentLocation`
- `Results!*.Value.PrecursorResult.PeptideResult.DocumentLocation`

The first refers to a peptide without reference to a replicate, while the second refers to a peptide within a replicate. If you set the Skyline document location with a value from the second column, Skyline will select the peptide *and change the active replicate*.

The values returned from these columns are strings. To turn them back into DocumentLocation objects, use the DocumentLocation.Parse method. ExampleInteractiveTool does that when the user clicks on a bar in the peak area graph:

```
// Select the peptide in Skyline when the user clicks on it.
var documentLocation = DocumentLocation.Parse(
    _selectedReplicate == "All" ? _peptideLinks[e.Index] : _replicateLinks[e.Index]);
_toolClient.SetDocumentLocation(documentLocation);
```

Document location

You can get and set the selection in the Skyline Targets view, or "document location", using the following methods:

- `DocumentLocation GetDocumentLocation()`
- `string GetDocumentLocationName()`
- `void SetDocumentLocation(DocumentLocation documentLocation)`

GetDocumentLocation gets the current Skyline document location. This can be used as an argument for other methods, like GetChromatograms.

GetDocumentLocationName returns the text of the selected element in the Skyline Targets view. In case of a multiple selection, this will be the element with the dotted focus rectangle. ExampleInteractiveTool uses this function to set the title of its chromatogram graph.

SetDocumentLocation changes the selection in the Skyline Targets view. As mentioned above, if the DocumentLocation argument specifies a replicate, Skyline will change the active replicate in addition to changing the Targets view selection.

Getting the replicate name

You can get the name of the active replicate in Skyline using this method:

- `string GetReplicateName()`

For such simple functions, you might wonder why this is not just a property. We consciously avoided using properties because they are normally evaluated automatically in the debugger. Each of these methods initiates cross-process communication, and you do not want that to happen automatically when stepping through code in the debugger.

Getting chromatograms

You can retrieve chromatograms for a given DocumentLocation using this method:

- `Chromatogram[] GetChromatograms(DocumentLocation documentLocation)`

Chromatogram data will be returned for a particular replicate, if the argument specifies one. Otherwise, chromatograms will be returned for all replicates.

Here is how ExampleInteractiveTool retrieves chromatograms for the active replicate when the Skyline selection changes:

```
var documentLocation = _toolClient.GetDocumentLocation();
var chromatograms = _toolClient.GetChromatograms(documentLocation);
```

The Chromatogram structures that are returned are as follows:

```
[Serializable]
public class Chromatogram
{
    public double PrecursorMz { get; set; }
    public double ProductMz { get; set; }
    public float[] Times { get; set; }
    public float[] Intensities { get; set; }
    public Color Color { get; set; }
}
```

Each chromatogram has precursor and product mass-to-charge ratios, arrays of matching retention times and intensities, and the color Skyline is using to display the chromatogram. If your tool displays information about the chromatogram, sometimes it can be handy to use the same color as Skyline to achieve a stronger visual correlation.

Getting the document path

To get the path to the open document in Skyline, use the following function:

- `string GetDocumentPath()`

Getting Skyline version

If your tool has a dependency on a particular version of Skyline, you can check that:

- `Version GetSkylineVersion()`

The Version structure is implemented as follows:

```
[Serializable]
public class Version
{
    public int Major { get; set; }
    public int Minor { get; set; }
    public int Build { get; set; }
    public int Revision { get; set; }
}
```

Register for document and selection events

Your tool can register for Skyline document and selection change events like this:

```
_toolClient.DocumentChanged += OnDocumentChanged;
_toolClient.SelectionChanged += OnSelectionChanged;
```

ExampleInteractiveTool performs registration when its main window is constructed, and unregisters when the main window is closed:

```
protected override void OnClosed(EventArgs e)
{
    base.OnClosed(e);

    try
    {
        _toolClient.DocumentChanged -= OnDocumentChanged;
        _toolClient.SelectionChanged -= OnSelectionChanged;
        _toolClient.Dispose();
    }
    catch
    {
    }

    _toolClient = null;
}
```

The code for the event handlers is shown below:

```
/// <summary>
/// Recreate graph when the Skyline document changes.
/// </summary>
private void OnDocumentChanged(object sender, EventArgs eventArgs)
{
    // Create graph on UI thread.
    Invoke(new Action(CreateGraph));
}

/// <summary>
/// Change the chromatogram graph when the selection changes.
/// </summary>
private void OnSelectionChanged(object sender, EventArgs eventArgs)
{
    // Create graph on UI thread.
    var documentLocation = _toolClient.GetDocumentLocation();
    var chromatograms = _toolClient.GetChromatograms(documentLocation);
    var documentLocationName = _toolClient.GetDocumentLocationName();
    Invoke(new Action(() =>
        _chromatogramGraph.CreateChromatograms(
            chromatograms, documentLocationName)));
}
```

Future plans

The first release of Skyline Interactive Tool Support is still somewhat limited. Through reports, tools can access most of the values Skyline uses internally, but aside from changing the selection and replicate, tools cannot change state within Skyline.

We expect that to change.

Skyline Interactive Tool Support is built on an architecture explicitly designed to be extensible. We would like to hear your ideas for tools that might need additional support. We are willing to extend the tool service to enable your ideas.