Intro Guide

Semantic Merge

The Ultimate Merging Machine

Contents

Semantic Merge tool

The Semantic Merge tool is a language-dependent source code merge tool. It can make a huge number of merge scenarios really easy – particularly ones that can't be managed by current text-based, language-agnostic, merge tools.

Since it is language-dependent, there will be different releases of Semantic Merge specific for different programming languages.

We have started with C# and Visual Basic, with Java joining soon and C++ to follow. Further languages will be chosen based on user feedback: JavaScript, Objective-C, Ruby? It's your call. :-)

Our Semantic Merge tool leverages the current merge technology of the merge tool included in Plastic SCM – which is already capable of dealing with refactors through Xmerge (http://plasticscm.com/features/xmerge.aspx) – and also the merge system of the Plastic SCM server itself, and combines them together with language-dependent parsing to create the ultimate source code merging machine.

Semantic Merge is not limited to Plastic SCM and can be configured to work with Git, Subversion, Perforce, ClearCase, Team Foundation Server, Mercurial, and many others.

The merge problem

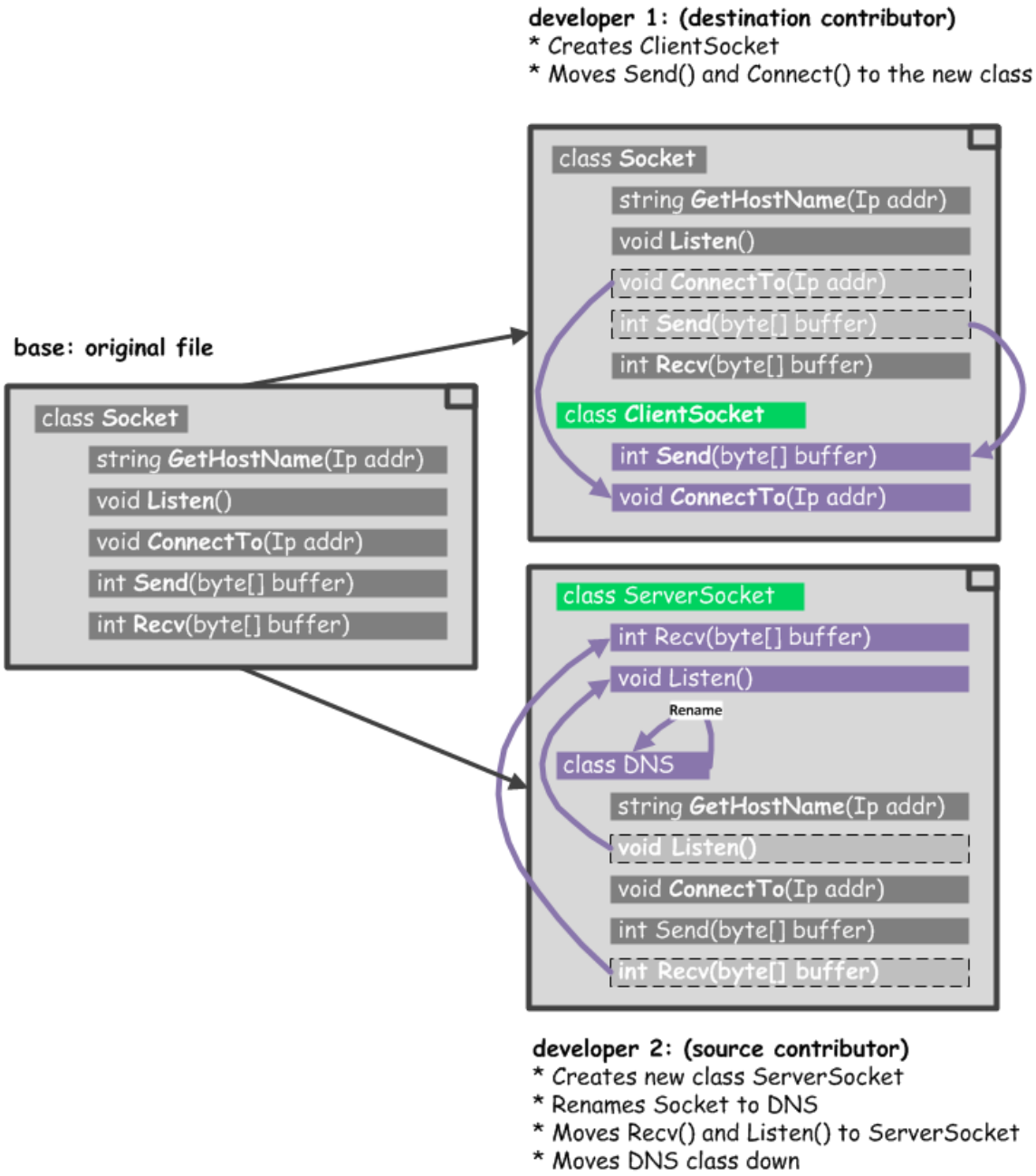
These days, software development is based on the "modify-merge" working pattern: developers work in parallel on the same codebase performing concurrent changes – potentially inside the same files – that will need to be reconciled back ("merged" in version control terms). In order to perform the merge, developers rely on "merge tools".

All merge systems use text based algorithms. The tools won't actually consider the programming language the code is written in, only the modifications made to the text. This way, all merge tools are language-unaware, and hence, they all have a wide operation range.

Not being able to act based on the specific programming language structures means that the merge tools are heavily dependent on the position of the texts being modified, which severely restricts developers' ability to perform changes concurrently and improve code quality and readability by refactoring.

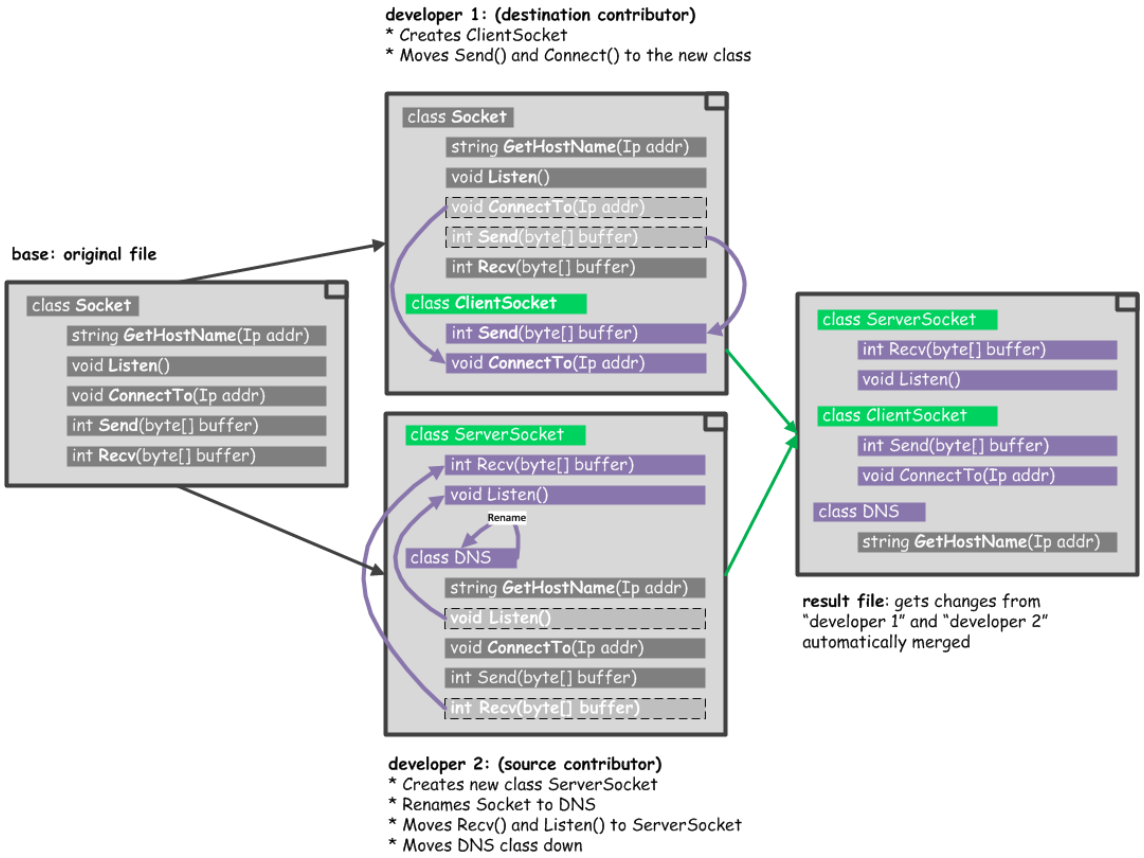
Language aware merge tool

But, how would a merge tool with programming language support behave? Suppose the following scenario:



This merge would be a nightmare for any merge tool on the market, but at the end of the day there's not a single conflict, if you look at it from a programmer's point of view. You just made some refactors in parallel, that's all.

So you'd expect the following result:



And this is exactly what we get with Semantic Merge!

Since it looks into the code structure and doesn't use a textual comparison method to compare the three contributors (it is a 3-way merge after all) the algorithm is not restricted by the relative positions of the texts it compares. Learn more about 2-way vs 3-way merge here: http://codicesoftware.blogspot.com/2010/11/live-to-merge-merge-to-live.html

Our tool parses the code, checks the obtained structures, and merges based on the "code trees" of the base plus the 3 contributors, automatically providing the expected result.

Enter Semantic Merge – class splitting

Let's check the Socket refactor example again with actual code, using the Semantic Merge tool. These are the three files to merge (the same example as above, but now with the actual code):

Base file

```
using System.Net;

namespace Network
{
    internal class Socket
```

```
    {
        internal string GetHostByName(string addr)
        {
            // this method returns the host
            // when you give an IP
            return CalculateHostByName(addr);
        }

        internal void Listen()
        {
            // do the listen on a port
            // and whatever it is needed
            // to listen
        }

        internal void ConnectTo(string addr)
        {
            // connect to a client
            Net.ConnectTo(addr);
        }

        internal int Send(byte[] buffer)
        {
            System.IO.Write(buffer);
        }

        internal int Recv(byte[] buffer)
        {
            System.IO.Read(buffer);
        }
    }
}
```

Source contributor (change by developer 2)

```
using System.Net;

namespace Network
{
    internal class Socket
    {
        internal string GetHostByName(string addr)
        {
            // this method returns the host
            // when you give an addr
            return CalculateHostByName(addr);
        }

        internal void Listen()
        {
            // do the listen on a port
            // and whatever it is needed
            // to listen
        }

        internal int Recv(byte[] buffer)
        {
            System.IO.Read(buffer);
        }
    }
```

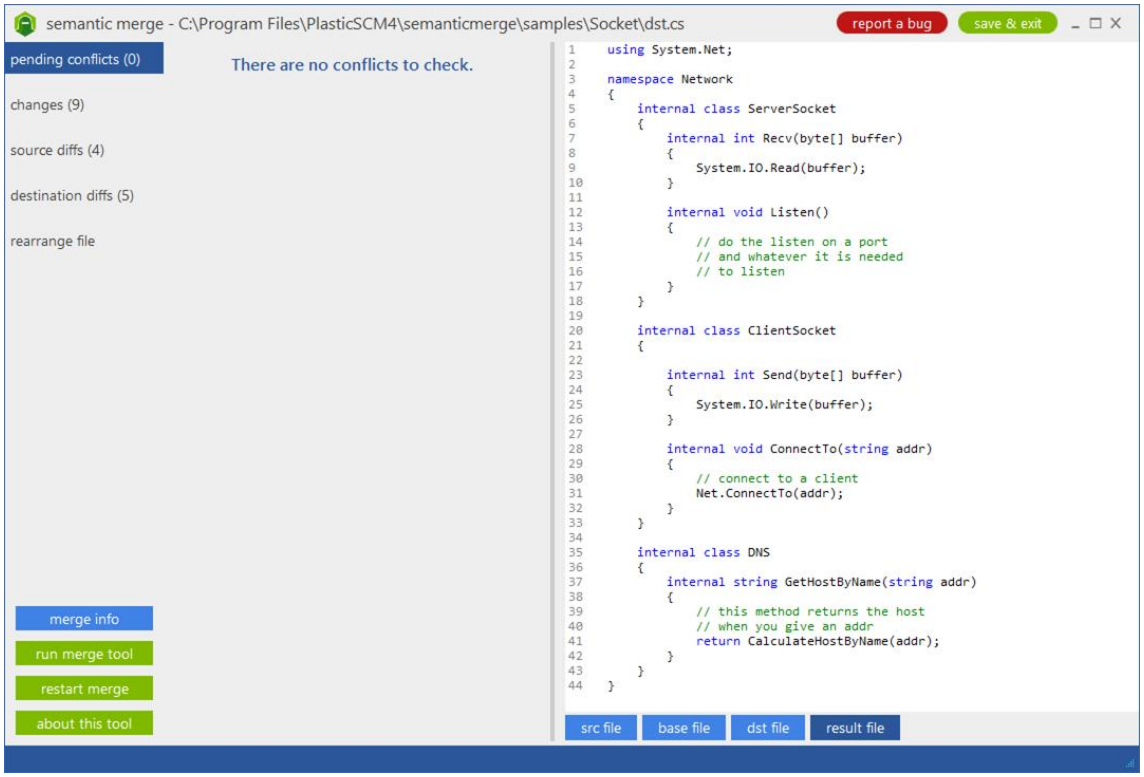
```
    internal class ClientSocket
    {
        internal int Send(byte[] buffer)
        {
            System.IO.Write(buffer);
        }

        internal void ConnectTo(string addr)
        {
            // connect to a client
            Net.ConnectTo(addr);
        }
    }
}
```

Destination contributor (change by developer 1)

```
using System.Net;

namespace Network
{
    internal class ServerSocket
    {
        internal int Recv(byte[] buffer)
        {
            System.IO.Read(buffer);
        }

        internal void Listen()
        {
            // do the listen on a port
            // and whatever it is needed
            // to listen
        }
    }

    internal class DNS
    {
        internal string GetHostByName(string addr)
        {
            // this method returns the host
            // when you give an IP
            return CalculateHostByName(addr);
        }

        internal void ConnectTo(string addr)
        {
            // connect to a client
            Net.ConnectTo(addr);
        }

        internal int Send(byte[] buffer)
        {
            System.IO.Write(buffer);
        }
    }
}
```

Semantic Merge output

Here is the result Semantic Merge provides, given the three files above:

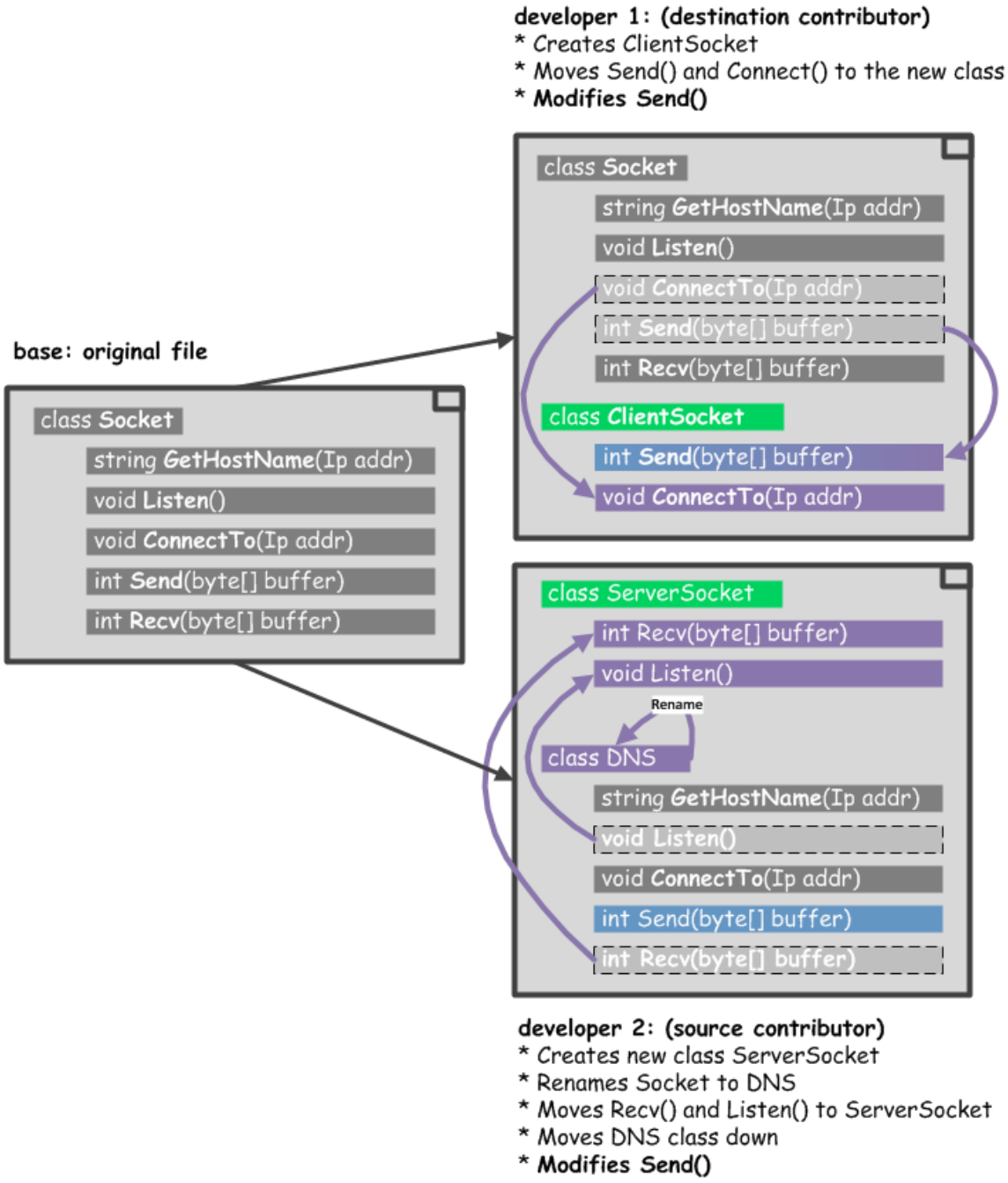


As you can see, there are no pending conflicts to be resolved and the result file looks exactly how you expect.

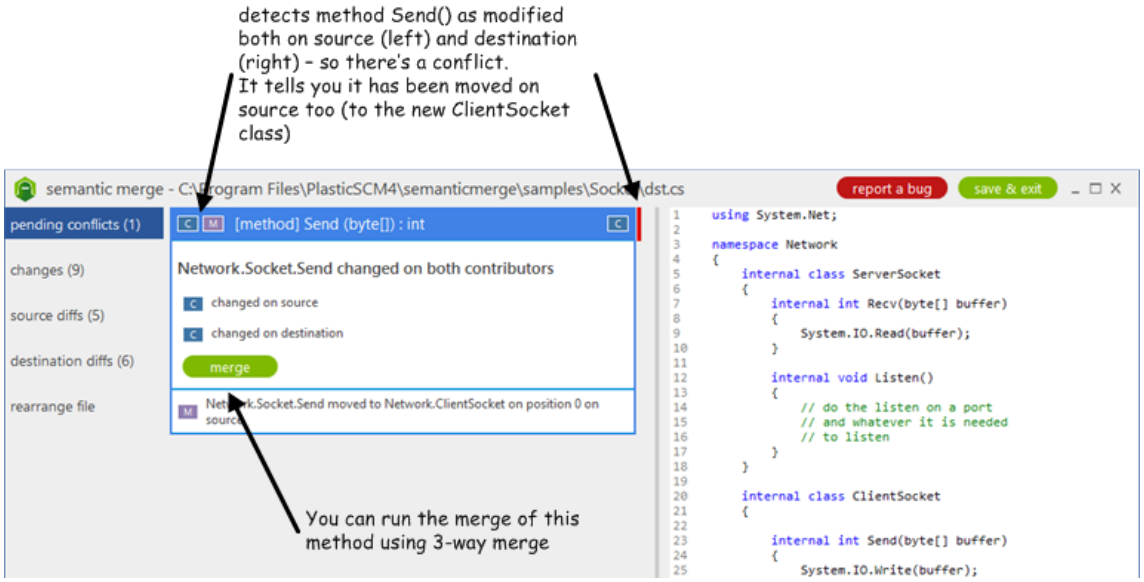
Slightly more complex merge – manual conflict

Let's consider again the scenario of the Socket class that is split into three classes. What if the two developers decide to modify the Send() method?

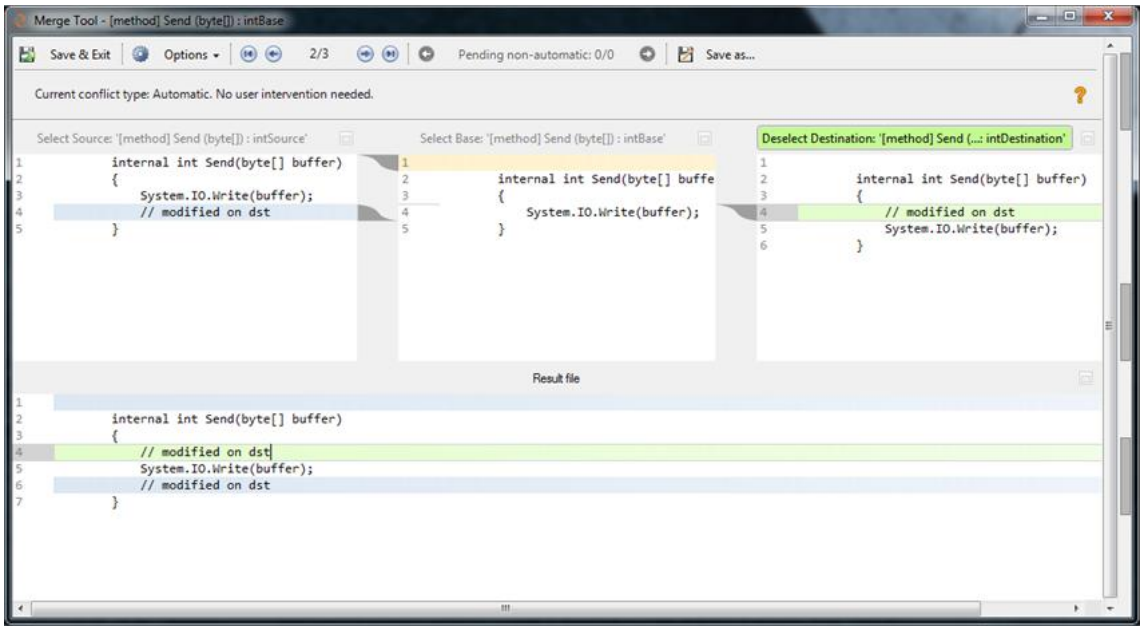
Source Contributor would modify Send() on its final location inside ClientSocket, while Destination Contributor would get the Send() modified inside the renamed (and moved) DNS class:



Let's see how Semantic Merge detects the conflict:



As you can see, the tool only detects 1 conflict (try to do the same with a conventional text-based merge tool and you'll enter into nightmare mode). You can run your favorite 3-way merge tool to solve the merge (by default, Semantic Merge comes with its own 3-way mergetool, the one included in Plastic SCM that is able to do Xmerge https://www.plasticscm.com/features/xmerge.aspx):



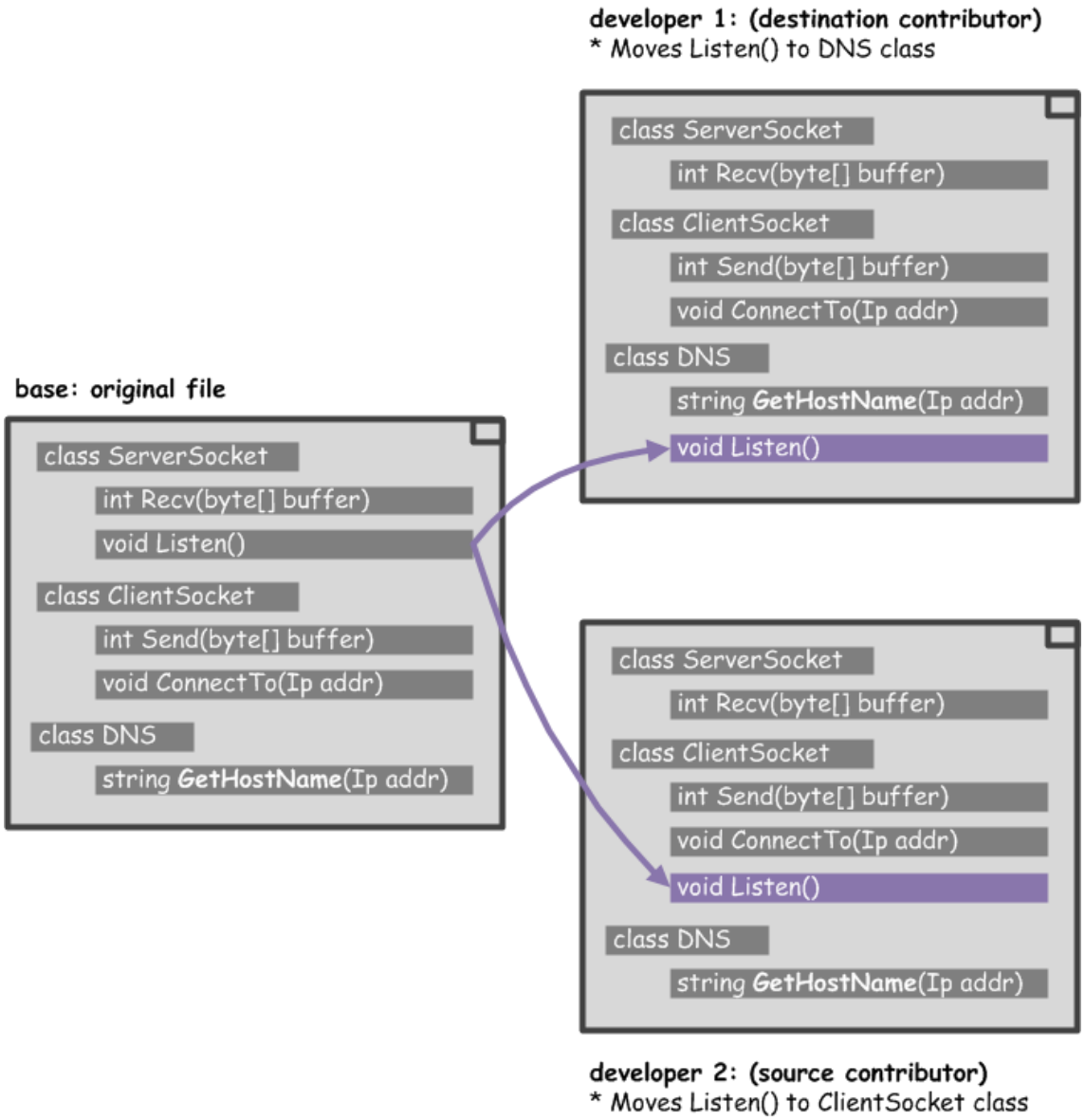
In my example, the merge has been fully automatic for the method too, since I didn't modify the same part of the method.

The advantage, as you can see, is that you use the divide and conquer method: you merge method by method (if needed) instead of the entire file, and the semantic merge is able to detect the classes, methods, properties, etc. independently of their locations; it can track them when they've been moved, renamed, and more.

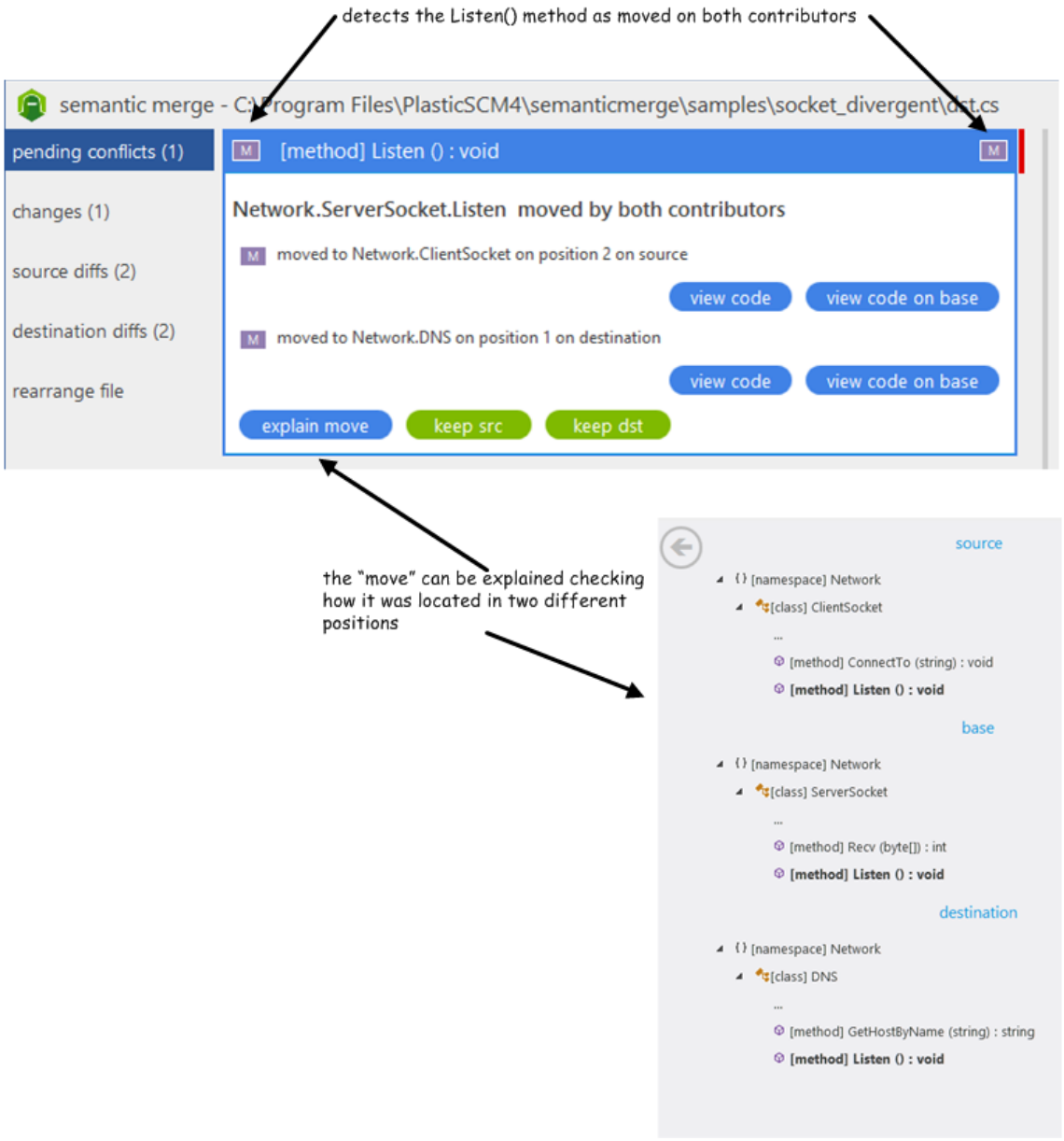
Going deeper – dealing with a divergent move

After the previous merge cycle, now suppose we take the resulting file as the base for the next iteration.

The two developers now decide to move the Listen() method, but each of them to a different location, as the following image shows:

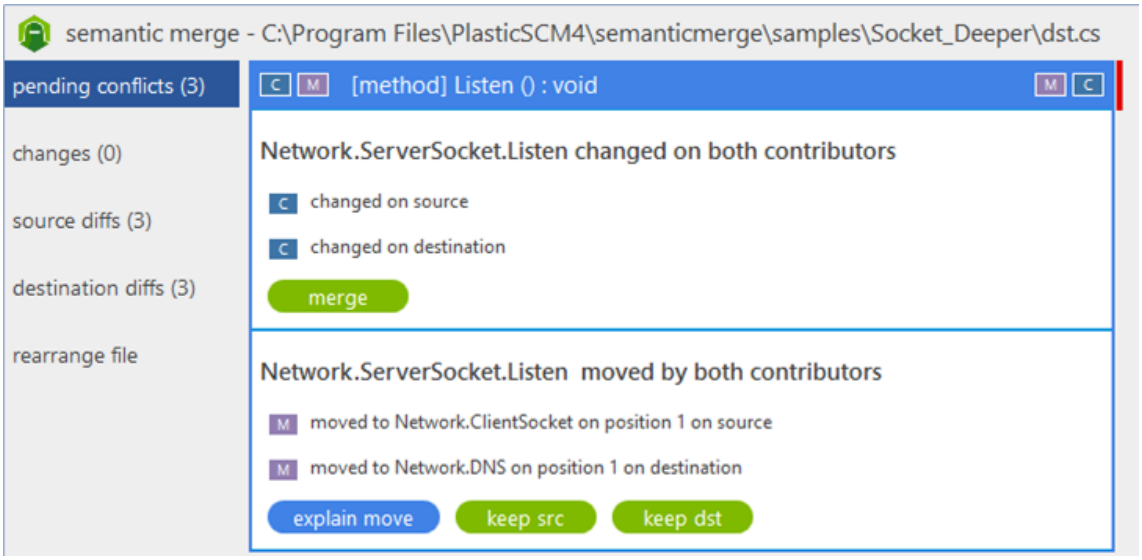


This is what the Semantic Merge tool detects as a "divergent move" and it will be handled as follows:



The developer running the merge can choose whether he wants to keep the method on the source location, the destination, or even duplicate it, keeping both contributors.

What if the two developers not only moved the Send() method but also modified it? Let's see how the tool handles the case:

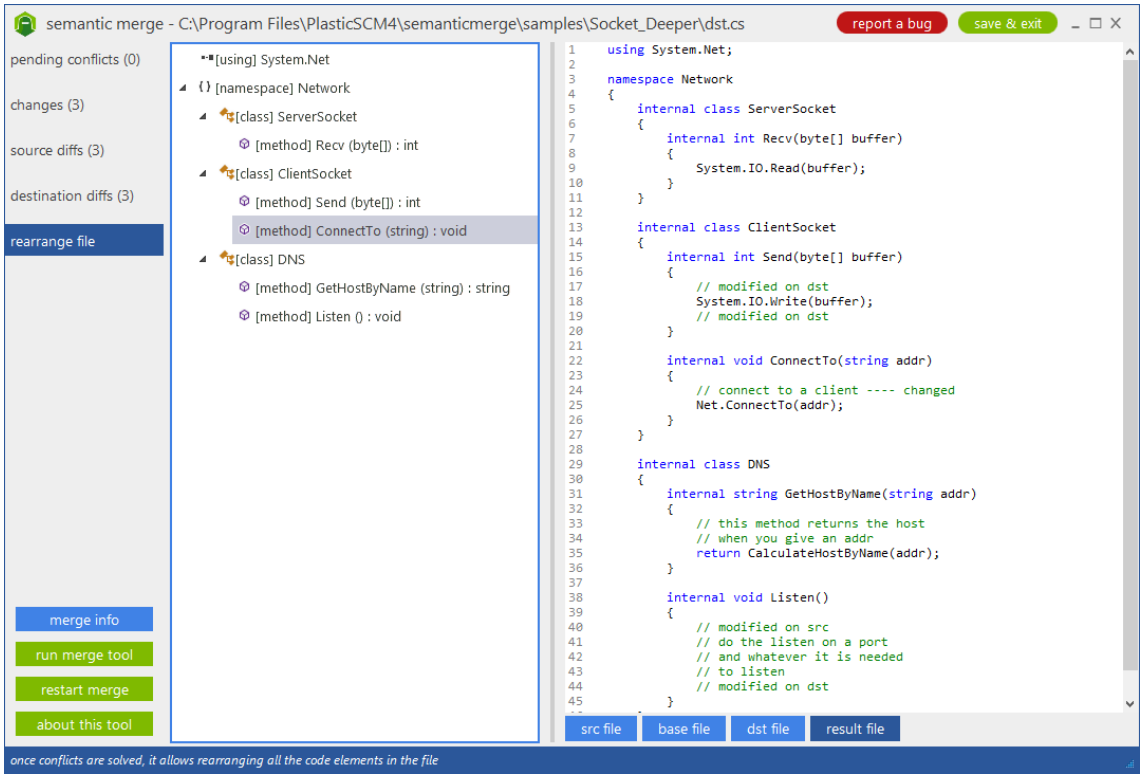


As you can see there is a "double conflict" on the method: you first have to resolve the changed/changed conflict and then the moved/moved.

In either case, it makes easy a situation that would be close to impossible to deal with using traditional text-based merge tools.

Rearrange the file after merging

Once you're done solving the conflicts, you can use the rearrange option to change the location of all elements inside the file. The changes will be immediately reflected on the result file:



Similarly, if you directly edit the result file with the editor on the right, the rearrange tree will be updated accordingly.

More merging capabilities

You guessed it right: Semantic Merge is able to "understand" the code structure and hence there are many situations where it can be a great aid:

- **Suppose you always want to review conflicts if one method is modified in parallel:** Text-based merge tools can detect when a block of text has been modified in parallel, but if you modified the first line of a method and someone else modified the last part, the merge will be automatic, even if there are potential logic issues. This is easily handled by Semantic Merge.
- **"Usings" (or "imports" in Java jargon) are also handled by the system:** If you add using System.Text on the first line and I add it on the fifth, Semantic Merge knows it is the same "using" so it will only add it once.
- **Changed/deleted:** Suppose you modified a method inside a subclass and I go and delete the class. Semantic Merge will deal with this specific case.
- The same holds true for many other scenarios like moved/moved, added/moved, and so on.
- What if I modify two methods and you go and decide to rearrange the class based on visibility rules? Public goes first, then internal, protected, and finally private. It will be an automatic, easy merge for Semantic Merge.

If we listed all the merge cases specifically handled by Semantic Merge, we'd be here all day. This short list, however, gives a pretty good idea of what the tool can do.