```
// --------------------------------
// Initialization:

// instance of IAnalysesRunner defines what analyses will be run
// There are two implementations at the moment: StaticAnalysesRunner and TypeAnalysisRunner
// AnalysesRunnerBase class should be subclassed when implementing new runners.
var runner = new StaticAnalysesRunner(
    x => { Console.WriteLine(x); },
    new TypeAnalysisSettings { WarningsAnalysis = true, AnnotateBasicBlocks = true });

// TablesContextManager provides context information for ITypeTable objects.
// There can be several ITypeTable objects that are scanned for referenced code elements.
// The basic implementation of ITypeTable uses the parsed source codes, but custom
// implementations can be provided. All the objects of type ITypeTable need to share the
// same instance of TablesContextManager.
var context = new TablesContextManager();

// Driver: manages the analysis process, it needs to know what analyses to
// run (via IAnalysesRunner) and we can also give it additional ITypeTable objects.
var driver = new AnalysisDriver(runner, context, new MyTables(context));

// For the common use cases, there are static factory methods in the AnalysisDriver class.
// The code above is equivalent to:
driver = AnalysisDriver.CreateStaticAnalysisDriver(
    x => { Console.WriteLine(x); },
    new TypeAnalysisSettings { WarningsAnalysis = true, AnnotateBasicBlocks = true },
    new WarningAnalysisSettings(),
    ctx => new MyTables(ctx));

// ----------------------
// Example1: add a file to the manager and analyzes all the code elements it the file
var file = driver.UpdateAndAnalyze("myfile.php", myFileAST);

// ----------------------
// Example2: add a file to the manager, but do not analyze anything yet. We will manually
// start analysis of global function named "main". This analysis may start analyses of
// other functions that "main" (transitively) references. Only adding a file to the manager
// (but not analyzing it) is useful when some function foo from another file references
// another function boo from our added (but not yet analyzed) file. If we start analyzing
// foo we will find out we need to analyze boo and because the manager already knows
// about boo, it can start its analysis.
var file2 = driver.UpdateAndAnalyze("myfile2.php", myFileAST2);

// the file object represents the code elements found in the file, for example:
```

```
// file.Functions - gives a list of all functions discovered in that file.
var func = file2.Functions.First(x => x.Name.Equals("main"));
driver.AnalyzeRoutine(new RoutineDecl(func));

// -----------------------
// Example3: We add a new statement to the "main" function and re-analyze it:
//
// Re-analysis is possible only when we set AnnotateBasicBlocks = true in the
// TypeAnalysisSettings. If we do this, the type analyzer will add some more
// additional data into the basic block (nodes of the control flow graph).
// These additional data take up some considerable amount of memory
// (64b * number of variables used in the function), but they allow fast re-analyzing.
// Re-analyzing is also not as precise as the proper data flow analysis, but has
// guaranteed complexity of O(n) where n is the number of statements we analyze.

var routineCtx = func.GetRoutineContext();
// We should know where we want to put the statement, let us say we want to
// put it after the second statement in the second basic block
var addedData = driver.Reanalyze(
    new RoutineDecl(func),
    routineCtx.FlowGraph.Blocks[2],
    /*lastStmtToAnalyzeIndex:*/ 2,
    myAdditionalStatement,
    routineCtx);

var classes = myAdditionalStatement.Expression.GetTypeInfo().GetTypes(routineCtx.TypeInfoCor

// Adding a new statement adds a new elements into the RoutineContext,
// namely it may contain new local variables, new types that were not referenced
// anywhere else in the function and possibly other things. To avoid
// unnecessary growth of the RoutineContext, we get instance of RoutineContext.AddedData
// class and when we are done with the re-analyzing (and processing the results of the
// re-analysis), we can call RoutineContext.AddedData.RemoveAddedData.
// Note: added data must be removed in the same order as they were added.
addedData.RemoveAddedData();
```