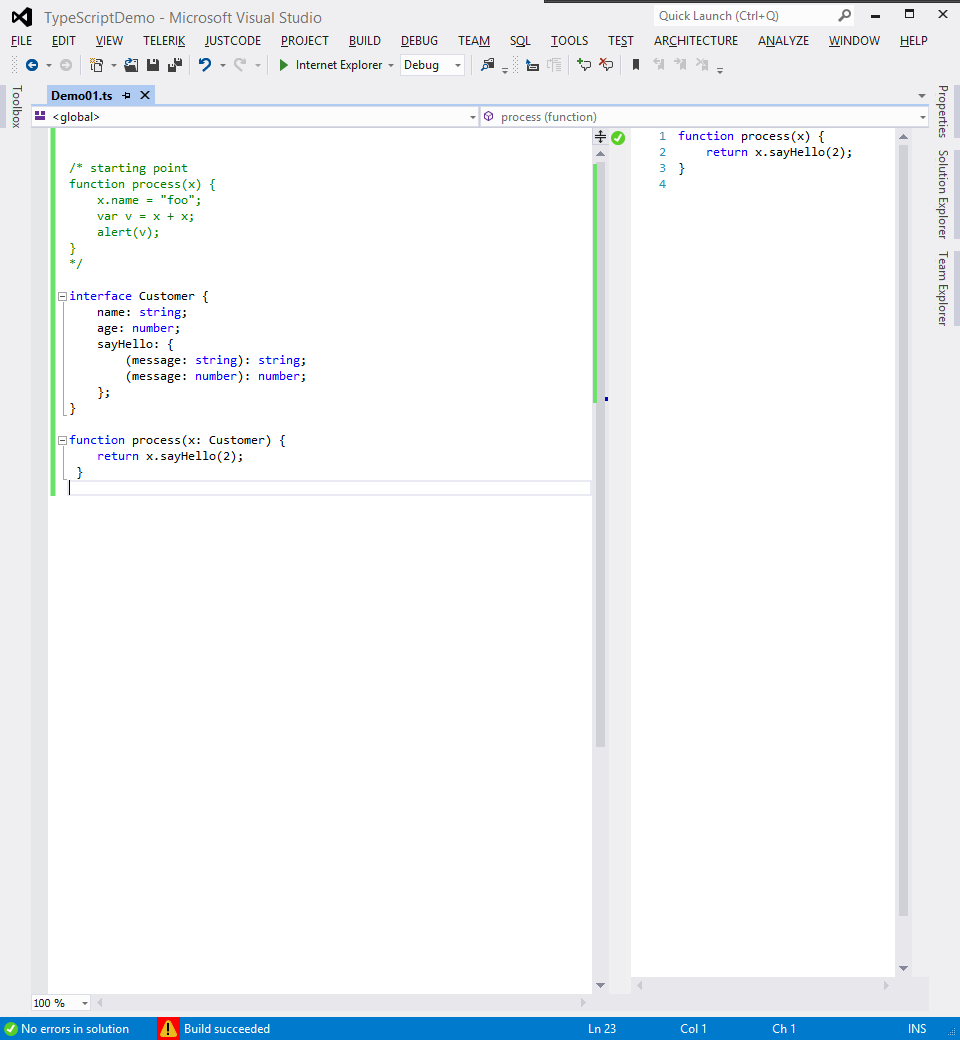
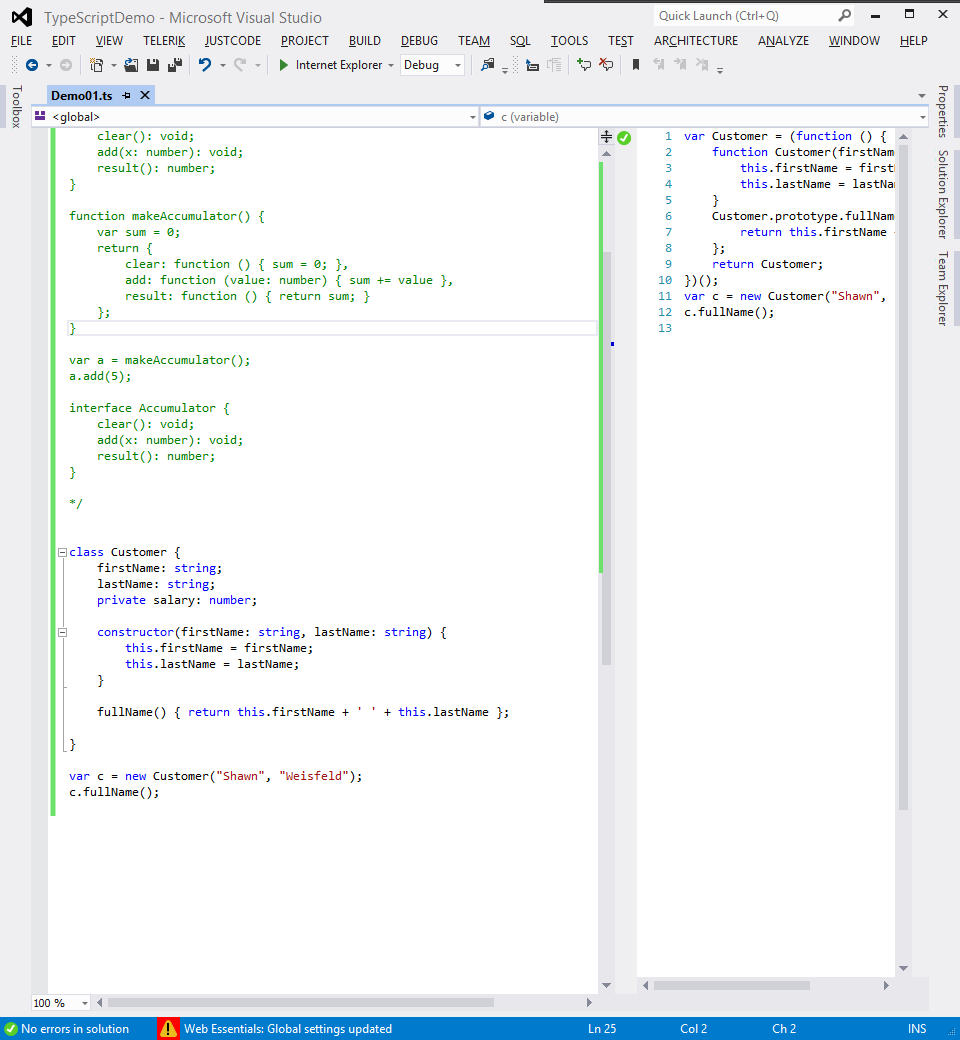
1. Demo 1 – TypeScript type system
   1. Show starting code
   2. Just plain javascript
   3. But makes no sense
   4. By default x is of type **any**, TypeScript is inferring this from its usage
   5. What if you intended for x to be a string?
   6. Just tell TypeScript your intent (add “: string” annotation to x declaration)
   7. Notice the squigglie on the name property, strings don’t have a name property (remove that line of code)
   8. Change the type of x to number
   9. Notice that we can infer the type of v from the fact that we are adding 2 numbers together
   10. Notice that we now have a squigglie on alert, since alert is expecting to receive a string (remove the alert line)
   11. Now lets try the last of the primitive types, bool
   12. Notice that we cannot add Booleans
   13. So while you “could” do those things in JavaScript they don’t make much sense, TypeScript allows you to focus your code down to the stuff that makes sense (remove the addition line)
   14. We can do Structured Types ( change the type of x to string[])
   15. (add a line x[0]) You get all the methods for a string in the array
   16. We can also say that x is a function that returns a string (change the type to () => string)
   17. Now if we call x() you see that the return is a string
   18. We can also say x is an object that has a name and an age (change the type to {name: string; age: number;})
   19. Show the properties on x
   20. Change the method to return x.name, show that the function has inferred its return type
   21. We could also name our type (copy the type to make an interface call customer, update process to take a customer)
   22. Now lets call the process method (var c = process({name: "Shawn", age: 10});)
   23. Show that process takes a customer an returns a string
   24. And TypeScript can infer the type of string
   25. Add a bool isRetired property to customer, now we get an error on our call to process as it is missing that flag
   26. We can tell TypeScript that isRetired is optional (add a ? before the : ) and you can see that the compile error goes away
   27. Show that it is still there by dotting into x
   28. Add the property to the object passed to the process method show that it works
   29. Show that if you pass a number to it instead of a bool you get your squiggles back
   30. Delete the method call, and the isRetired property and add a function (sayHello(message: string): string;)
   31. Change the call in process to exercise the sayHello method, show that the return type of the process method is still being inferred by TypeScript
   32. Add a new argument to the sayHello function (, message2: string) show that we get the squiggles, back, but we can also make that argument optional by adding the optional argument
   33. We can also overload sayHello by passing in numbers (show the overloads on the call to sayHello, and show that TypeScript can infer the number return type when the call the number overload)
   34. Show the compiled version, since it is not possible to “enforce” it at runtime there is only one method call, however the TypeScript rules keep everything safe during development.
   35. You could have written the overload like this, with 2 call signatures



* 1. I could also add a property, foo of type any, to sayHello (show that it exists in intellisense)
  2. You can also add a constructor ( new (s: string): Element; ) show what it looks like in intellisense (hint: return new x( )
  3. You can also add a indexer ( [index: number]: Date; ) show what it looks like in intellisense (hint: return x[0]. )
  4. makeAccumulator example
  5. Show that TypeScript has inferred all three methods on the function, show the intellisense on the object.
  6. Add the interface code, and implement it on the function
  7. Change one of the methods, show that you get a compile error since the function no longer implements the interface
  8. This is also helpful when implementing event handlers
  9. Add call to window.onmousemove = function(e) { e. and show the intellisense, and that e is a mouseevent
  10. do a go to definition onmousemove, show that there is a big interface file for everything in javascript

1. Demo 2 – code structure with typescript



* 1. Create a customer
  2. Give it a first and last name properties
  3. Give it a constructor
  4. Add a full name method
  5. Convert the full name method to a property (add the word get before the function name, show the squigglie, and the change in the javascript)
  6. Add age Private member, show that it affects intellisense, but the real javascript is the same as all other properties, mention that although it is private in the TypeScript but there is no javascript analog.
  7. Change properties to automatic properties
     1. Remove the property and the setting in the constructor
     2. Add public to the arguments on the constructor
     3. Show that you can add a default value
     4. Show that you can remove them from the constructor
  8. Create a derived class (extends keyword)
     1. Show the generated helper code
     2. Show the generated constructor
     3. Copy the constructor from the base class, add a field
     4. Overwrite the base fullname function, call super.full name to get to the base class implementation
  9. Arrow functions
     1. Lexically scoped this
  10. Internal Modules
      1. Export
      2. Modules are open ended
      3. Deep nesting
         1. Import
  11. External Modules

1. TypeScript Compiler written in TypeScript