```
        int pageLength = -1;
        //start from 1st record
        int start = 1;
        int noOfPages;
        //call domain method
        Collection<Customer> customersList
        = customers.FindAll(LoadStatus.Loaded,1,pageLength,
         out noOfPages);
        customerArray = new CustomerDTO[customersList.Count];
        for(int i=0; i<customersList.Count; i++)
        {
            customerArray[i] = customersList[i].DTO;
        }
        return customerArray;
    }
```

The above web method returns a list of of all the customers from the database. The `GetAllCustomers()` method uses the domain method `FindCustomers()` and converts the strongly-typed collection of `Customer` objects returned into an array of `CustomerDTO`s. The highlights of this very simple method are:

- The method returns a list of data transfer objects instead of core domain objects. This helps us stop the domain model from being exposed to external clients and also helps us include loose coupling in our system.
- The web method uses the domain model internally to get a list of objects to be returned to the outside caller. So the interface layer abstracts the business layer by creating wrappers around it, and external clients (consuming our service interface) do not need to know anything about domain model internals, as long as they know which web methods to call and with what parameters.
- This web method can be used in any external client irrespective of the platform on which the client is running. This is possible because web services use the XML format which is generic across all platforms. So an application running on a JAVA platform can call our method and use it internally.

- This method has no arguments, but it is calling a domain method internally:
  ```
  customers.FindAll(LoadStatus.Loaded,1,pageLength,
        out noOfPages);
  ```
  which takes `LoadStatus`, start page, `pageLength` and number of pages as arguments. We have specified `pageLength` as 1 so that we can get all of the records. This is a very simple example of how we can wrap fine-grained functionality with coarse-grained methods to be exposed to external clients. It is up to the consuming client to handle the returned data, and page it as per the client's own external logic. Our aim should be to simply return as much data as we can in one trip, and avoid making multiple trips across the network (which can be performance-intensive). Because we do not know what logic external clients might have in their own domain model, we have to keep our service interface as simple and generic as possible, and leave the data manipulation and logic to the clients themselves.

We know that DTOs are serializable (marked with the `serialize` attribute). But if we simply returned the generic list of DTOs, we would face two problems:

- We would need to serialize the generic list. This is an easy task and does not require complex coding.
- Only clients based on NET 2.0 or above would be able to use the web service, as generic objects are a .NET 2.0-only feature. So having a generic list in the method signature is not a good idea.

To avoid these two issues, we should always try for simple data types in web service method signatures. That is the reason we have used arrays in the above sample method.

Consuming Services

Now we will see how we can use this service in our web forms UI instead of using business objects directly. If you recall, in Chapter 4, our `showAll.aspx` form had displayed a list of all of the customers using the code shown here (taken from the `showAll.aspx.cs` code-behind file in Chapter 4, a 5-tier source code):

```
/// <summary>
   /// Get and Show a list of all customers
   /// </summary>
   private void FillAllCustomers()
   {
       CustomerCollection customers = new CustomerCollection();
       int noOfPages;
       int pageLength = -1;
       int start = 1;
```