Consider the following **Put**() method. This method updates the specified Employee details.

public HttpResponseMessage Put(int id, Employee employee)

{

    try

    {

        using (EmployeeDBEntities entities = new EmployeeDBEntities())

        {

            var entity = entities.Employees.FirstOrDefault(e => e.ID == id);

            if (entity == null)

            {

                return Request.CreateErrorResponse(HttpStatusCode.NotFound,

                    "Employee with Id " + id.ToString() + " not found to update");

            }

            else

            {

                entity.FirstName = employee.FirstName;

                entity.LastName = employee.LastName;

                entity.Gender = employee.Gender;

                entity.Salary = employee.Salary;

                entities.SaveChanges();

                return Request.CreateResponse(HttpStatusCode.OK, entity);

            }

        }

    }

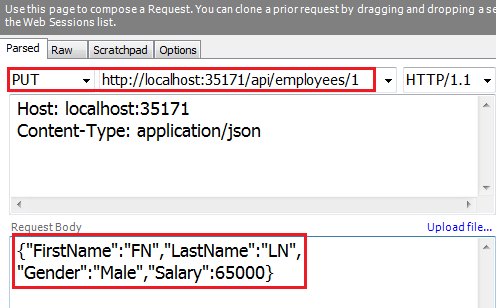
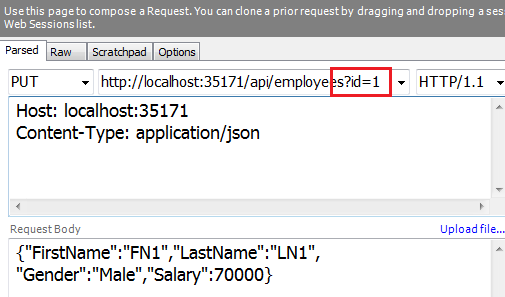
    catch (Exception ex)

    {

        return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);

    }

}

To update employee details whose Id is 1 we issue a Put request to the following URI  
http://localhost/api/employees/1  
  
If you are using Fiddler, the PUT request is as shown below. Notice the Id of the employee is in the URI and the employee data is in the request body.   
   
  
At this point if we execute the request, the employee data is updated as expected.   
  
Now let's include the Id as a query string parameter. In the first request Id is specified as part of route data. Notice in Fiddler we have included id parameter as a query string.   
   
  
When we execute this request, the update succeeds as expected.  
  
When a PUT request is issued, Web API maps the data in the request to the PUT method parameters in the EmployeesController. This process is called Parameter Binding.  
  
Now let us understand the default convention used by Web API for binding parameters.

1. If the parameter is a simple type like int, bool, double, etc., Web API tries to get the value from the URI (Either from route data or Query String)
2. If the parameter is a complex type like Customer, Employee etc., Web API tries to get the value from the request body

So in our case, the id parameter is a simple type, so Web API tries to get the value from the request URI. The employee parameter is a complex type, so Web API gets the value from the request body.   
  
We can change this default **parameter binding** process by using [FromBody] and [FromUri] attributes. Notice in the example below

1. We have decorated id parameter with [FromBody] attribute, this forces Web API to get it from the request body
2. We have decorated employee parameter with [FromUri] attribute, this forces Web API to get employee data from the URI (i.e Route data or Query String)

public HttpResponseMessage Put([FromBody]int id, [FromUri]Employee employee)

{

    try

    {

        using (EmployeeDBEntities entities = new EmployeeDBEntities())

        {

            var entity = entities.Employees.FirstOrDefault(e => e.ID == id);

            if (entity == null)

            {

                return Request.CreateErrorResponse(HttpStatusCode.NotFound,

                    "Employee with Id " + id.ToString() + " not found to update");

            }

            else

            {

                entity.FirstName = employee.FirstName;

                entity.LastName = employee.LastName;

                entity.Gender = employee.Gender;

                entity.Salary = employee.Salary;

                entities.SaveChanges();

                return Request.CreateResponse(HttpStatusCode.OK, entity);

            }

        }

    }

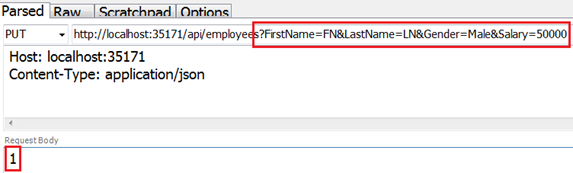
    catch (Exception ex)

    {

        return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);

    }

}

**Here is the request from Fiddler**  
1. Employee data is specified in the URI using query string parameters   
2. The id is specified in the request body    
  
  
When we execute the request the update succeeds as expected