Beispiel / Service

Interface Definition Language (Protobuf)

```
syntax = "proto3";
option csharp_namespace = "BasicExample";
package Greet;

// Service definition
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest)
    returns (HelloReply);
}

// Request message containing the user's name
message HelloRequest {
  string name = 1;
}

// Response message containing the greetings
message HelloReply {
  string message = 1;
}
```

Service-Definition

Service-Methode mit Request / Response

Message Type

Service Implementation

Generierte Basisklasse

```
public class GreeterService : Greeter.GreeterBase
{

    public override async Task<HelloReply> SayHello(
        HelloRequest request,
        ServerCallContext context)
    {
        return await Task.FromResult(new HelloReply
        {
            Message = "Hello " + request.Name
        });
    }
}
```

Überschr. Methode

Implementation

OST

Beispiel / Client

Service Client

```
// The port number(5001) must match the port of the gRPC server.
GrpcChannel channel = GrpcChannel.ForAddress("https://localhost:5001");
Greeter.GreeterClient client = new Greeter.GreeterClient(channel);

HelloReply reply = await client.SayHelloAsync(
    new HelloRequest { Name = "GreeterClient" }
);

Console.WriteLine("Greeting: " + reply.Message);
```

Generierter Client Stub

Remote Procedure Call

OST

Proto Files

- Datei-Endung *.proto
- Header
  - Allgemeine Definitionen (syntax, option, etc.)
- Services
  - 0 oder mehr Services
  - 1 oder mehr Service-Methoden pro Service
- Message Types
  - 1 oder mehr Fields
  - Field definiert sich aus
    - Type
    - Unique Name
    - Unique Field Number (Versionierung)

```
syntax = "proto3";
option csharp_namespace = "_01_BasicExample";
package Greet;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest)
    returns (HelloReply);
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings.
message HelloReply {
  string message = 1;
}
```

Proto Files

- Service-Methoden haben immer genau
  - 1 Parameter
  - 1 Rückgabewert

- Null-Werte / Leere Message

```
import "google/protobuf/empty.proto";
...
google.protobuf.Empty
```

```
syntax = "proto3";
option csharp_namespace = "_01_BasicExample";
package Greet;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest)
    returns (HelloReply);
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings.
message HelloReply {
  string message = 1;
}
```

Messages / Fields

- Angabe des Feldtypen
  - Skalarer Werttyp
  - Anderer Message Type
  - Enumeration
- Unique Field Name
  - Wird für Generatoren verwendet
  - Lower Snake Case (Underscores)
- Unique Field Number
  - Identifikator für das Binärformat
  - Wertebereich 1 bis 536’870’911
    Ausnahme: 19’000 bis 19’999*

```
message SearchRequest {
  string query = 1;
  int32 page_number = 2;
  int32 result_per_page = 3;
}
```

* vom Protokoll reserviert

OST

Fields / Repeated Fields

- Zwei Arten von Fields
  - Singular Default / Skalarer Wert
  - Repeated Liste von Werten

- Schlüsselwort “repeated”
- Ergibt eine Liste von Strings

```
message SearchResponse {
  repeated string results = 1;
}
```


OST

Enumerations

- Von der Idee her analog zu Enumerationstypen (enum) in .NET
- Definition
  - Innerhalb einer Message
  - Proto-File Root
- Enum-Member mit dem Wert 0 muss zwingend existieren
  - Wird als Default Value verwendet
- Schlüsselwort “reserved” kann auch für Enumerations verwendet werden

```
message SearchRequest {
  Color searchColor = 1;
  Size searchSize = 2;

  enum Color {
    RED = 0; // 0 must exist
    GREEN = 1;
  }
}

enum Size {
  S = 0; // 0 must exist
  M = 1;
  L = 2;
}
```


OST

Message Type Composition & Imports

- Message Types können ebenfalls als Field verwendet werden
- Import eines *.proto Files über das “import” Schlüsselwort

```
// File: example.proto
import "protos/_base.proto";

message Search {
  Query query = 1;
  LogicalOperator operator = 2;
}
message Query {
  string filter = 1;
}

// File: _base.proto
enum LogicalOperator {
  AND = 0;
  OR = 1;
  XOR = 2;
}
```


OST

Reserved Fields

- Für Versionierung gedacht
- Wiederverwendung wird vom Protocol Buffer Compiler verhindert
- Schlüsselwort “reserved” bei
  - Unique Field Name
  - Unique Field Number

- Ranges können mit “to” reserviert werden
  `reserved 1 to 3`

```
message SearchRequest {
  reserved 1, 3, 20 to 30;
  reserved "page_number",
           "result_per_page";

  string query = 1; // Compilerfehler
  int32 page_number = 2; // Compilerfehler
  int32 result_per_page = 3; // Compilerfehler
}
```


OST

Generierter Code

- Services müssen beim Startup registriert werden
  - Startup.cs (app.UseEndpoints)
    `endpoints.MapGrpcService<...>();`
  - Sonst
    `Grpc.Core.RpcException: 'Status(StatusCode=Unimplemented, Detail="Service is unimplemented.")'`
- Generierte Methoden aus abstrakter Basisklasse müssen Implementiert werden
  - Sonst
    `Grpc.Core.RpcException: 'Status(StatusCode=Unimplemented, Detail="")'`

```
public class GreeterService : Greeter.GreeterBase
{
    public override async Task<HelloReply> SayHello(
        HelloRequest request,
        ServerCallContext context)
    {
        return await Task.FromResult(new HelloReply
        {
            Message = "Hello " + request.Name
        });
    }
}
```


OST

C# API / Startup

- Registrierung der gRPC Types via Dependency Injection

```
public class Startup
{
    public void ConfigureServices(
        IServiceCollection services)
    {
        services.AddGrpc();
    }

    public void Configure(
        IApplicationBuilder app,
        IWebHostEnvironment env)
    {
        // ...
        app.UseEndpoints(ep =>
        {
            ep.MapGrpcService<MyCustomerService>();
            ep.MapGrpcService<MyOrderService>();
            // ...
        });
        // ...
    }
}
```

- Definition der Endpoints
  - Einmal pro Service

OST

Beispiel Customer Service

Proto-File

CustomerService

```
service CustomerService {
  rpc GetCustomers (google.protobuf.Empty)
      returns (GetCustomersResponse);

  rpc GetCustomer (GetCustomerRequest)
      returns (GetCustomerResponse);
}
```

OrderService

```
service OrderService {
  rpc GetOrders (GetOrdersRequest)
      returns (GetOrdersResponse);
}
```

Beispiel Customer Service

Proto-File

Messages (Customer)

```
message GetCustomersResponse {
  repeated CustomerResponse data = 1;
}
message GetCustomerResponse {
  CustomerResponse data = 1;
}

message GetCustomerRequest {
  int32 id_filter = 1;
  bool include_orders = 2;
}
message CustomerResponse {
  int32 id = 1;
  string first_name = 2;
  string last_name = 3;
  Gender gender = 4;
  repeated OrderResponse orders = 10;
}
enum Gender { UNKNOWN = 0; FEMALE = 1; MALE = 2; }
```

Messages (Order)

```
message GetOrdersRequest {
  int32 customer_id_filter = 1;
}
message GetOrdersResponse {
  repeated OrderResponse data = 1;
}
message OrderResponse {
  string product_name = 1;
  int32 quantity = 2;
  double price = 3;
}
```

Beispiel Customer Service

Service-Implementation

CustomerService

```
public class MyCustomerService
    : CustomerService.CustomerServiceBase
{
    public override async
        Task<GetCustomersResponse>
        GetCustomers(
            Empty request,
            ServerCallContext context)
    { /* ... */ }

    public override async
        Task<GetCustomerResponse>
        GetCustomer(
            GetCustomerRequest request,
            ServerCallContext context)
    { /* ... */ }
}
```

OrderService

```
public class MyOrderService
    : OrderService.OrderServiceBase
{
    public override async
        Task<GetOrdersResponse>
        GetOrders(
        GetOrdersRequest request,
        ServerCallContext context)
    { /* ... */ }
}
```

Beispiel Customer Service

Client-Implementation (Customer)

```
// The port number (5001) must match the port of the gRPC server.
GrpcChannel channel = GrpcChannel.ForAddress("https://localhost:5001");

// Customer service calls
var customerClient = new CustomerService.CustomerServiceClient(channel);

var request1 = new Empty();
GetCustomersResponse response1 = await customerClient.GetCustomersAsync(request1);
Console.WriteLine(response1);

var request2 = new GetCustomerRequest { IdFilter = 1 };
GetCustomerResponse response2 = await customerClient.GetCustomerAsync(request2);
Console.WriteLine(response2);

request2.IncludeOrders = false;
response2 = await customerClient.GetCustomerAsync(request2);
Console.WriteLine(response2);
```

Beispiel Customer Service

Client-Implementation (Order)

```
// The port number (5001) must match the port of the gRPC server.
GrpcChannel channel = GrpcChannel.ForAddress("https://localhost:5001");

// Order service calls
var orderClient = new OrderService.OrderServiceClient(channel);

var request3 = new GetOrdersRequest { CustomerIdFilter = 1 };
GetOrdersResponse response3 = await orderClient.GetOrdersAsync(request3);
Console.WriteLine(response3);
```

Protocol Buffers

- Schlüsselwort “stream” vor Typbezeichnung
  - Payload ist eine normale Message

- ReadFiles
  - Server Streaming Call (Server > Client)
- SendFiles
  - Client Streaming Call (Client > Server)
- RoundtripFiles
  - Bi-directional / Duplex Streaming Call

```
service FileStreamingService {
  rpc ReadFiles (google.protobuf.Empty)
      returns (stream FileDto);

  rpc SendFiles (stream FileDto)
      returns (google.protobuf.Empty);

  rpc RoundtripFiles (stream FileDto)
      returns (stream FileDto);
}

message FileDto {
   string file_name = 1;
   int32 line = 2;
   string content = 3;
}
```

Server Streaming Call | Server > Client

Client

Call Object (Wrapper)

No Parameters

Read last written chunk

```
using (AsyncServerStreamingCall<FileDto> call = client.ReadFiles(new Empty()))
{
    await foreach (FileDto message in call.ResponseStream.ReadAllAsync())
    {
        WriteLine($"File: {message.FileName}, Line Nr: {message.Line}, Line Content:
{message.Content}");
    }
}
```


OST

Server Streaming Call | Server > Client

Service

```
public override async Task ReadFiles(
    Empty request,
    IServerStreamWriter<FileDto> responseStream,
    ServerCallContext context)
{
    string[] files = Directory.GetFiles(@"...");
    foreach (string file in files)
    {
        string content; int line = 0;
        using StreamReader reader = File.OpenText(file);
        while ((content = await reader.ReadLineAsync()) != null)
        {
            line++;
            FileDto reply = new FileDto
            {
                FileName = file, Line = line, Content = content,
            };
            await responseStream.WriteAsync(reply);
        }
    }
}
```

No Parameters

Response Stream

File-Loop

Line-Loop

Write to Stream

OST

Client Streaming Call | Client > Server

Client

```
using (AsyncClientStreamingCall<FileDto, Empty> call = client.SendFiles())
{
    string[] files = Directory.GetFiles(@"Files");
    foreach (string file in files)
    {
        string content; int line = 0;
        using StreamReader reader = File.OpenText(file);
        while ((content = await reader.ReadLineAsync()) != null)
        {
            line++;
            FileDto reply = new FileDto
            {
                FileName = file, Line = line, Content = content,
            };
            await call.RequestStream.WriteAsync(reply);
        }
    }
    // Required!
    await call.RequestStream.CompleteAsync(); // No more messages to come (server exits foreach-Loop)
    Empty result = await call; // Wait until service method is terminated / Get the result
}
```

Call Object (Wrapper)

No Parameters

File-Loop

Line-Loop

Write to Stream

Close Stream / Call

Result (instance of Empty in this case)

OST

Client Streaming Call | Client > Server

Service

```
public override async Task<Empty> SendFiles(
    IAsyncStreamReader<FileDto> requestStream,
    ServerCallContext context)
{
    await foreach (FileDto message in requestStream.ReadAllAsync())
    {
        WriteLine(
            $"File: {message.FileName}, Line Nr: {message.Line}, Line Content: {message.Content}");
    }

    return new Empty();
}
```

Request Stream

Read last written chunk

Empty Result

OST

Bi-directional (Duplex) | Client > Server > Client

Client [1 of 2]

```
using (AsyncDuplexStreamingCall<FileDto, FileDto> call = client.RoundtripFiles())
{
    // Read
    Task readTask = Task.Run(async () =>
    {
        await foreach (FileDto message in call.ResponseStream.ReadAllAsync())
        {
            WriteLine(
               $"File: {message.FileName}, Line Nr: {message.Line}, Line Content: {message.Content}");
        }
    });

    // Write
    // ... See next Slide ...

    // Required!
    await call.RequestStream.CompleteAsync(); // No more messages to come (server exits foreach-Loop)
    await readTask; // Wait until service method is terminated / all messages are received by client
}
```

Call Object (Wrapper)

No Parameters

Read Task (no await)

Read last written chunk

Close Stream

Await Read Task

OST

Client [2 of 2]

```
    // Write
    string[] files = Directory.GetFiles(@"Files");
    foreach (string file in files)
    {
        string content; int line = 0;
        using StreamReader reader = File.OpenText(file);
        while ((content = await reader.ReadLineAsync()) != null)
        {
            line++;
            FileDto reply = new FileDto
            {
                FileName = file, Line = line, Content = content,
            };
            await call.RequestStream.WriteAsync(reply);
        }
    }

    // Required!
    await call.RequestStream.CompleteAsync(); // No more messages to come (server exits foreach-Loop)
    await readTask; // Wait until service method is terminated / all messages are received by client
}
```

File-Loop

Line-Loop

Write to Stream

OST

Bi-directional (Duplex) | Client > Server > Client

Service

```
public override async Task RoundtripFiles(
    IAsyncStreamReader<FileDto> requestStream,
    IServerStreamWriter<FileDto> responseStream,
    ServerCallContext context)
{
    await foreach (FileDto message in requestStream.ReadAllAsync())
    {
        await responseStream.WriteAsync(message);
        WriteLine(
            $"File: {message.FileName}, Line Nr: {message.Line}, Line Content: {message.Content}");
    }
}
```

Request Stream

Response Stream

Read last written chunk

Write to Stream

OST