Fallback strategy **CHEAT SHEET**

Basics

This reactive resilience strategy **allows you to define a substitute value** if the execution fail.

You can configure the behaviour of the strategy via the **FallbackStrategyOptions<T>** object.

Specify constant value for exception

```
new ResiliencePipelineBuilder<string>()
  .AddFallback(new FallbackStrategyOptions<string>
  {
    ShouldHandle = new PredicateBuilder<string>().Handle<Exception>(),
    FallbackAction = _ => Outcome.FromResultAsValueTask("exception fallback")
  })
```

Specify constant value for missing result

```
new ResiliencePipelineBuilder<string>()
  .AddFallback(new FallbackStrategyOptions<string>
  {
    ShouldHandle = new PredicateBuilder<string>().HandleResult(string.IsNullOrEmpty),
    FallbackAction = _ => Outcome.FromResultAsValueTask("missing data fallback")
  })
```

Specify value dynamically for inner exception

```
new ResiliencePipelineBuilder<string>()
  .AddFallback(new FallbackStrategyOptions<string>
  {
    ShouldHandle = new PredicateBuilder<string>().HandleInner<IOException>(),
    FallbackAction = async args =>
    {
      var fallbackValue = await GetFallbackValueAsync(args.Outcome.Exception!);
      return Outcome.FromResult(fallbackValue);
    }
  })
```

Specify asynchronously delegate for timeout notification

```
new ResiliencePipelineBuilder<string>()
  .AddFallback(new FallbackStrategyOptions<string>
  {
    ShouldHandle = new PredicateBuilder<string>().HandleResult(s => s.Length > 100),
    FallbackAction = args => Outcome.FromResultAsValueTask("too long fallback"),
    OnFallback = async args => await NotifyAsync(args.Outcome.Result!)
  })
```