VECTOR

SOME/IP with CANoe Communication Setup

Part 4. Simulation with CAPL Programming

V1.0 | 2023-07-18

Agenda

- **Overview**
- Application Model (CAPL)
- Technical References

This document is for …

- Help you Implement logic of SOME/IP application via CAPL Programming

- Help to understand for the Initial Setup
  - SOME/IP concept in CANoe
  - Import Data Sources
  - Import Application Models
  - Write Data Source as a vCDL
  - Simulation with CANoe GUI features
  - Analysis of SOME/IP data
  - Simulation with CAPL programming

covered in Part1, Part2, Part3

- CANoe sample configuration is prepared for your understanding
  - https://portal.vector.com/shared/068f1303-77de-4eab-8bd0-ca37938c6885

- Quick start guide for work of basic SOME/IP simulation, analysis
  - You can extend your knowledge and know-how through CANoe Help Manual!
  - It covers More detailed technical information.

Agenda

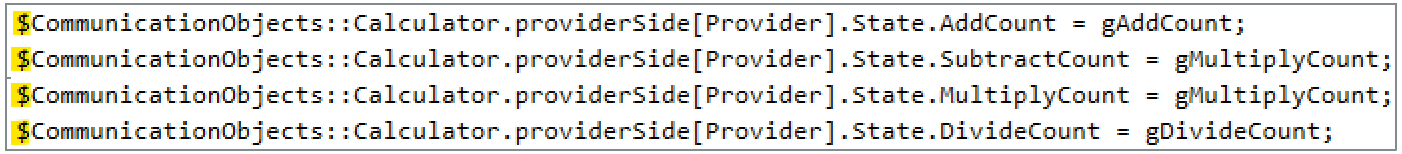
Overview

▶ **Application Model (CAPL)**

Technical References

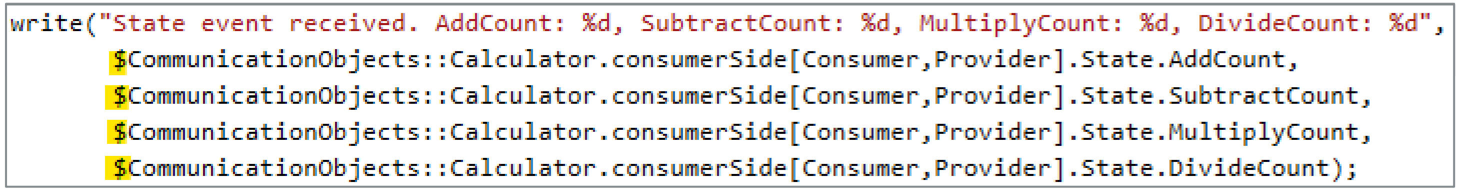
Basic CAPL Applications

- Value Access
  - Use $ sign for accessing communication values
    - ‵Write Value‵ can trigger Event and Field Notification



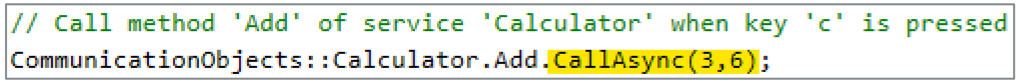
```
$CommunicationObjects::Calculator.providerSide[Provider].State.AddCount = gAddCount;
$CommunicationObjects::Calculator.providerSide[Provider].State.SubtractCount = gMultiplyCount;
$CommunicationObjects::Calculator.providerSide[Provider].State.MultiplyCount = gMultiplyCount;
$CommunicationObjects::Calculator.providerSide[Provider].State.DivideCount = gDivideCount;
```

    - ‵Read Value‵



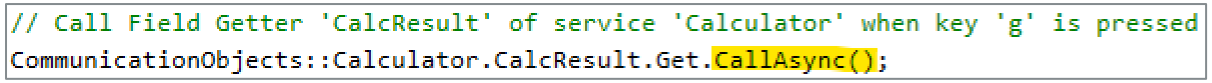
```
write("State event received. AddCount: %d, SubtractCount: %d, MultiplyCount: %d, DivideCount: %d",
       $CommunicationObjects::Calculator.consumerSide[Consumer,Provider].State.AddCount,
       $CommunicationObjects::Calculator.consumerSide[Consumer,Provider].State.SubtractCount,
       $CommunicationObjects::Calculator.consumerSide[Consumer,Provider].State.MultiplyCount,
       $CommunicationObjects::Calculator.consumerSide[Consumer,Provider].State.DivideCount);
```

- Request Call
  - Use CallAsync(params..) function for calling Method and Field Getter/Setter
    - ‵Method Call’

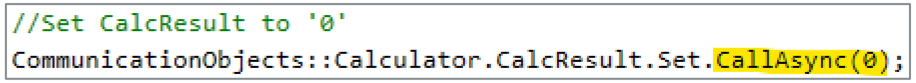


```
// Call method 'Add' of service 'Calculator' when key 'c' is pressed
CommunicationObjects::Calculator.Add.CallAsync(3,6);
```

    - ‵Field Getter/Setter’



```
// Call Field Getter 'CalcResult' of service 'Calculator' when key 'g' is pressed
CommunicationObjects::Calculator.CalcResult.Get.CallAsync();
```

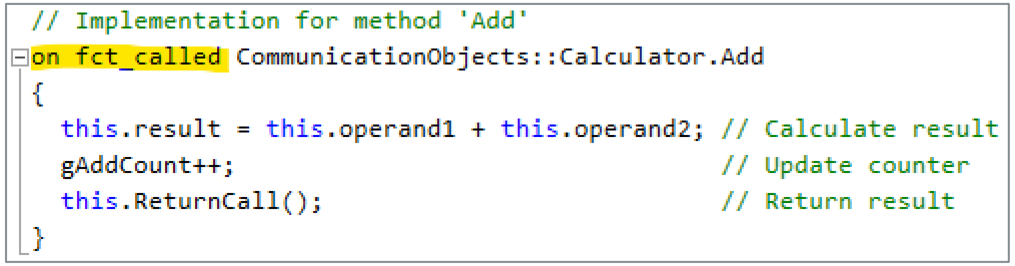


```
//Set CalcResult to '0'
CommunicationObjects::Calculator.CalcResult.Set.CallAsync(0);
```

  - The number and data type of input parameters follow the prototype defined by the Data Source(vCDL)

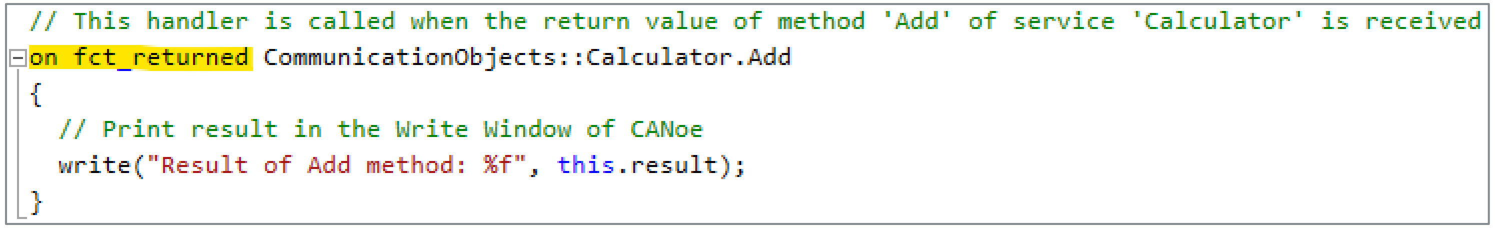
Basic CAPL Applications

- Event Handler
  - on fct_called / on fct_returned is called when Method and Field Getter/Setter are processed
    - Provider side - on fct_called is called when receive the request from consumer



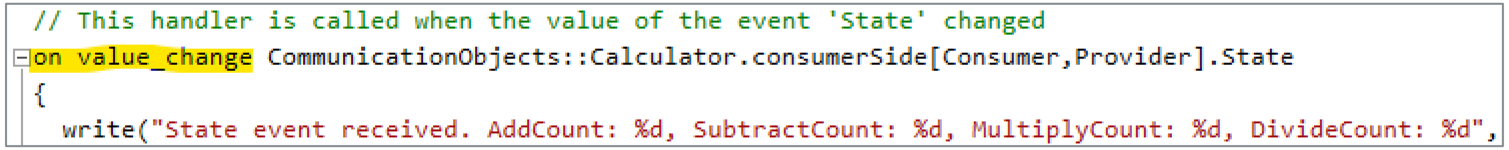
```
// Implementation for method 'Add'
on fct_called CommunicationObjects::Calculator.Add
{
  this.result = this.operand1 + this.operand2; // Calculate result
  gAddCount++;                                 // Update counter
  this.ReturnCall();                           // Return result
}
```

    - Consumer side – on fct_returned is called when receive the response from provider



```
// This handler is called when the return value of method 'Add' of service 'Calculator' is received
on fct_returned CommunicationObjects::Calculator.Add
{
  // Print result in the Write Window of CANoe
  write("Result of Add method: %f", this.result);
}
```

  - on value_change is called when value of Event / Field Notification changed

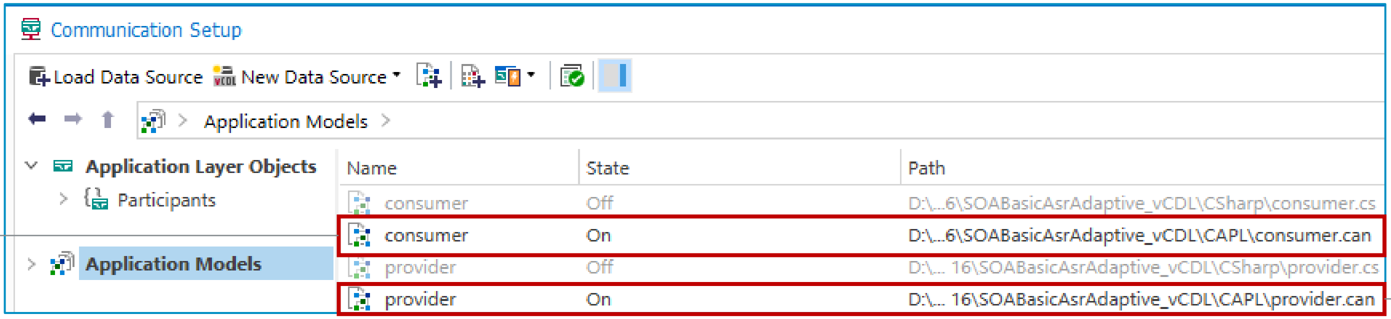


```
// This handler is called when the value of the event 'State' changed
on value_change CommunicationObjects::Calculator.consumerSide[Consumer,Provider].State
{
  write("State event received. AddCount: %d, SubtractCount: %d, MultiplyCount: %d, DivideCount: %d",
```

  - on value_update is called when value of Event / Field Notification updated

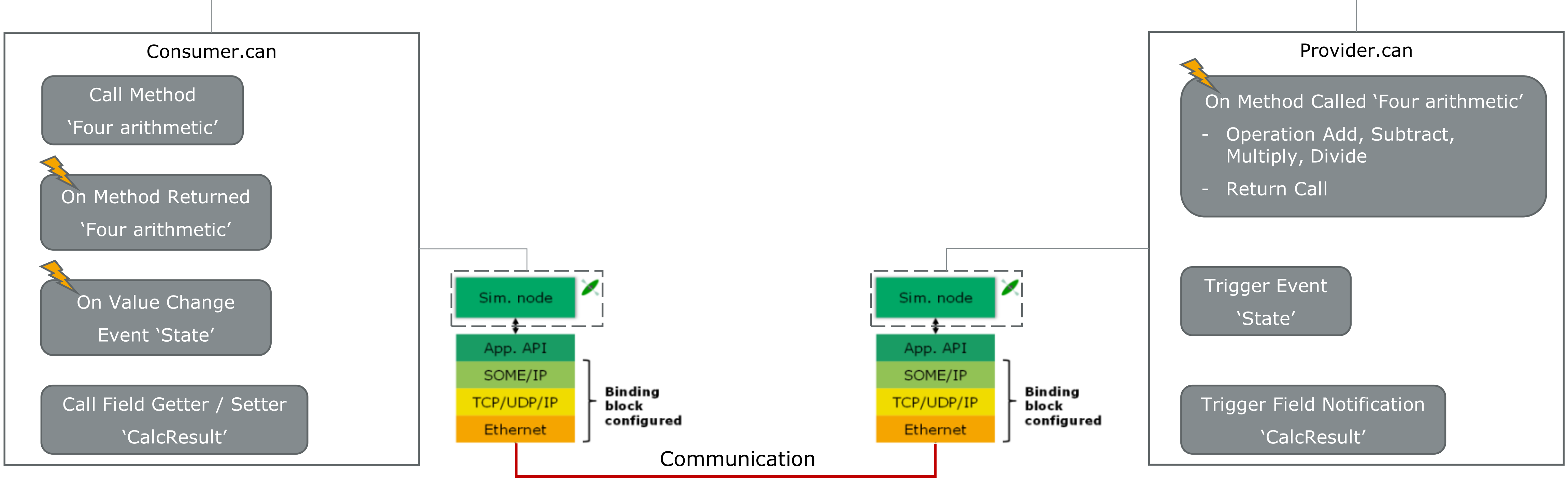
CAPL Practice

- Application Logics Example



Communication Setup

Name	State	Path
consumer	Off	D:\...6\SOABasicAsrAdaptive_vCDL\CSharp\consumer.cs
consumer	On	D:\...6\SOABasicAsrAdaptive_vCDL\CAPL\consumer.can
provider	Off	D:\... 16\SOABasicAsrAdaptive_vCDL\CSharp\provider.cs
provider	On	D:\... 16\SOABasicAsrAdaptive_vCDL\CAPL\provider.can



Consumer.can

- Call Method ‘Four arithmetic’
- On Method Returned ‘Four arithmetic’
- On Value Change Event ‘State’
- Call Field Getter / Setter ‘CalcResult’

Provider.can

- On Method Called ‘Four arithmetic’
  - Operation Add, Subtract, Multiply, Divide
  - Return Call
- Trigger Event ‘State’
- Trigger Field Notification ‘CalcResult’

CAPL Practice

- Scenario ‘Method Call‘

Consumer.can

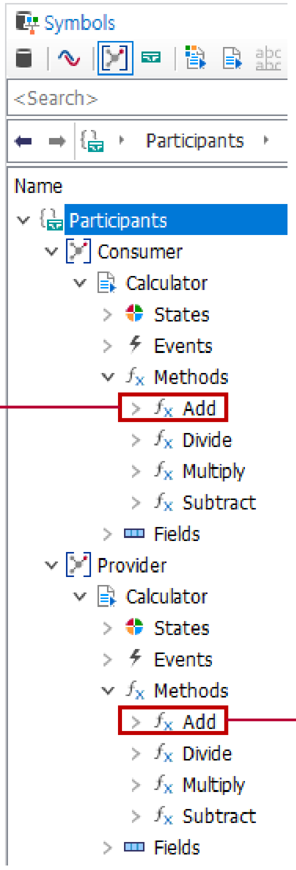
① Call Method ‘Add’ with 3,6

```
// Call method 'Add' of service 'Calculator' when key 'c' is pressed
on key 'c'
{
  CommunicationObjects::Calculator.Add.CallAsync(3,6);
}
```

Drag and Drop!

③ On Method Returned ‘Add’

```
// This handler is called when the return value of
// method 'Add' of service 'Calculator' is received
on fct_returned CommunicationObjects::Calculator.Add
{
  // Print result in the Write Window of CANoe
  write("Result of Add method: %f", this.result);
}
```



- Symbols Window -

Provider.can

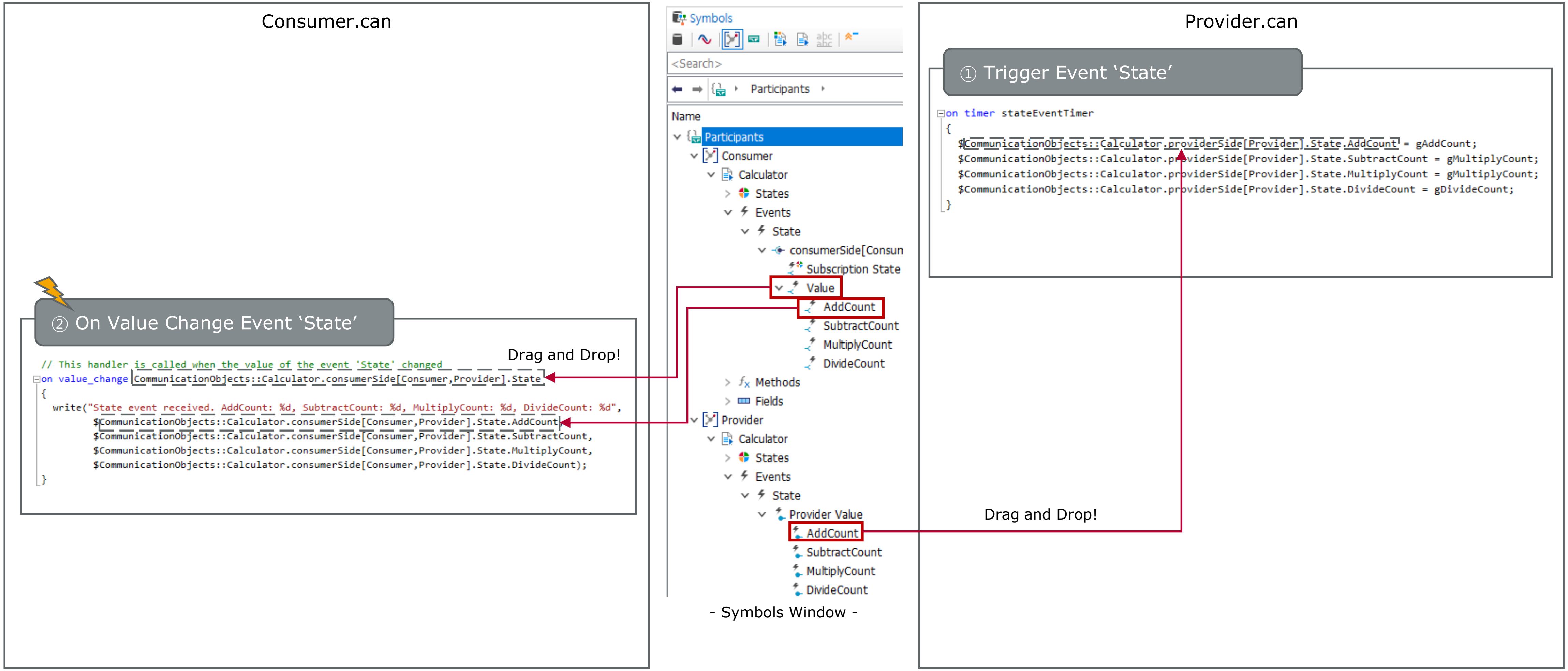
② On Method Called ‘Add’

```
// Implementation for method 'Add'
on fct_called CommunicationObjects::Calculator.Add
{
  this.result = this.operand1 + this.operand2; // Calculate result
  gAddCount++;                                 // Update counter
  this.ReturnCall();                           // Return result
}
```

Drag and Drop!

CAPL Practice

- Scenario ‘Event‘ Trigger



Consumer.can

② On Value Change Event ‘State’

Drag and Drop!

```
// This handler is called when the value of the event 'State' changed
on value_change CommunicationObjects::Calculator.consumerSide[Consumer,Provider].State
{
  write("State event received. AddCount: %d, SubtractCount: %d, MultiplyCount: %d, DivideCount: %d",
        $CommunicationObjects::Calculator.consumerSide[Consumer,Provider].State.AddCount,
        $CommunicationObjects::Calculator.consumerSide[Consumer,Provider].State.SubtractCount,
        $CommunicationObjects::Calculator.consumerSide[Consumer,Provider].State.MultiplyCount,
        $CommunicationObjects::Calculator.consumerSide[Consumer,Provider].State.DivideCount);
}
```

- Symbols Window -

Provider.can

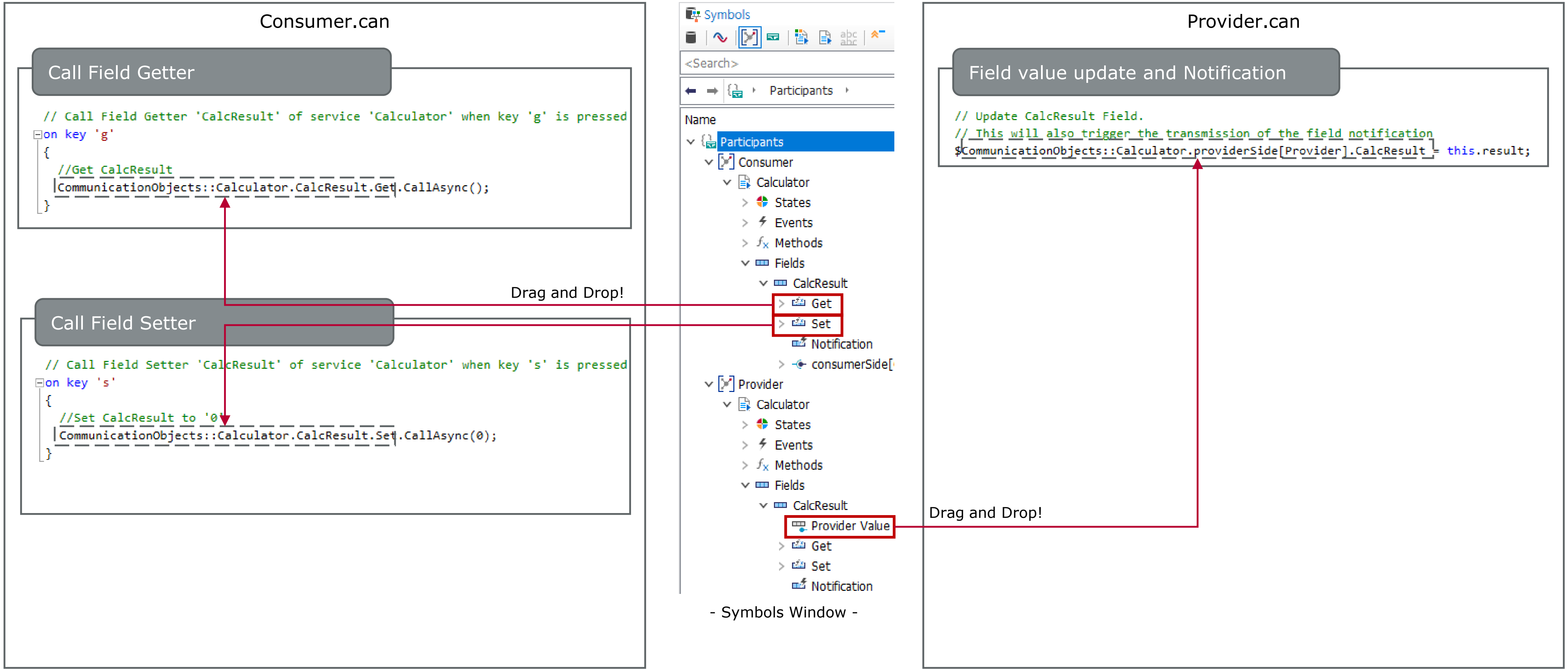
① Trigger Event ‘State’

```
on timer stateEventTimer
{
  $CommunicationObjects::Calculator.providerSide[Provider].State.AddCount = gAddCount;
  $CommunicationObjects::Calculator.providerSide[Provider].State.SubtractCount = gMultiplyCount;
  $CommunicationObjects::Calculator.providerSide[Provider].State.MultiplyCount = gMultiplyCount;
  $CommunicationObjects::Calculator.providerSide[Provider].State.DivideCount = gDivideCount;
}
```

Drag and Drop!

CAPL Practice

- Scenario ‘Field Notification‘ and ‘Field Getter and Setter‘



Consumer.can

Call Field Getter

```
// Call Field Getter 'CalcResult' of service 'Calculator' when key 'g' is pressed
on key 'g'
{
  //Get CalcResult
  CommunicationObjects::Calculator.CalcResult.Get.CallAsync();
}
```

Call Field Setter

```
// Call Field Setter 'CalcResult' of service 'Calculator' when key 's' is pressed
on key 's'
{
  //Set CalcResult to '0'
  CommunicationObjects::Calculator.CalcResult.Set.CallAsync(0);
}
```

Drag and Drop!

- Symbols Window -

Provider.can

Field value update and Notification

```
// Update CalcResult Field.
// This will also trigger the transmission of the field notification
$CommunicationObjects::Calculator.providerSide[Provider].CalcResult = this.result;
```

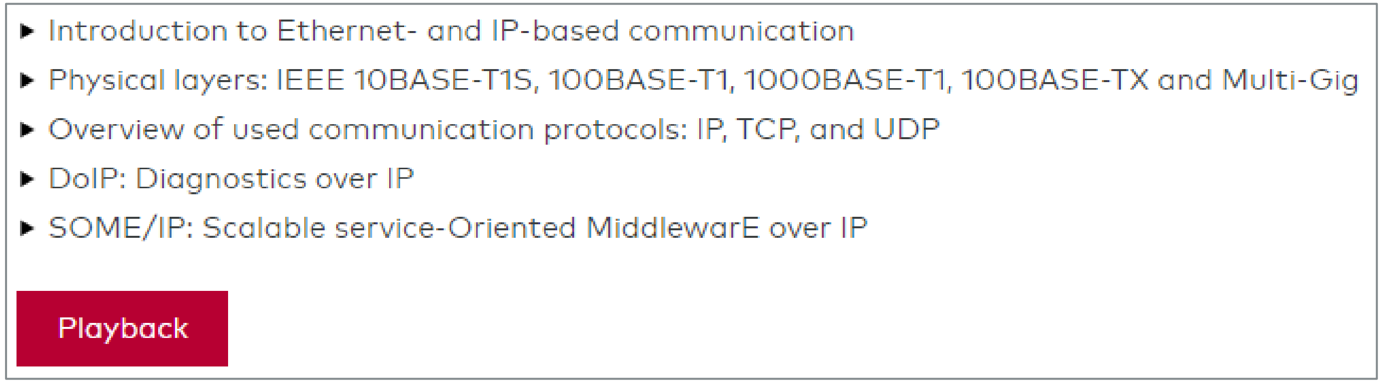
Drag and Drop!

Agenda

- Overview
- Application Model (CAPL)
- ▸ **Technical References**

Video Recording

- Introduction to Automotive Ethernet
  - https://www.vector.com/kr/ko/events/global-de-en/webinar-recordings/2023/introduction-to-automotive-ethernet/



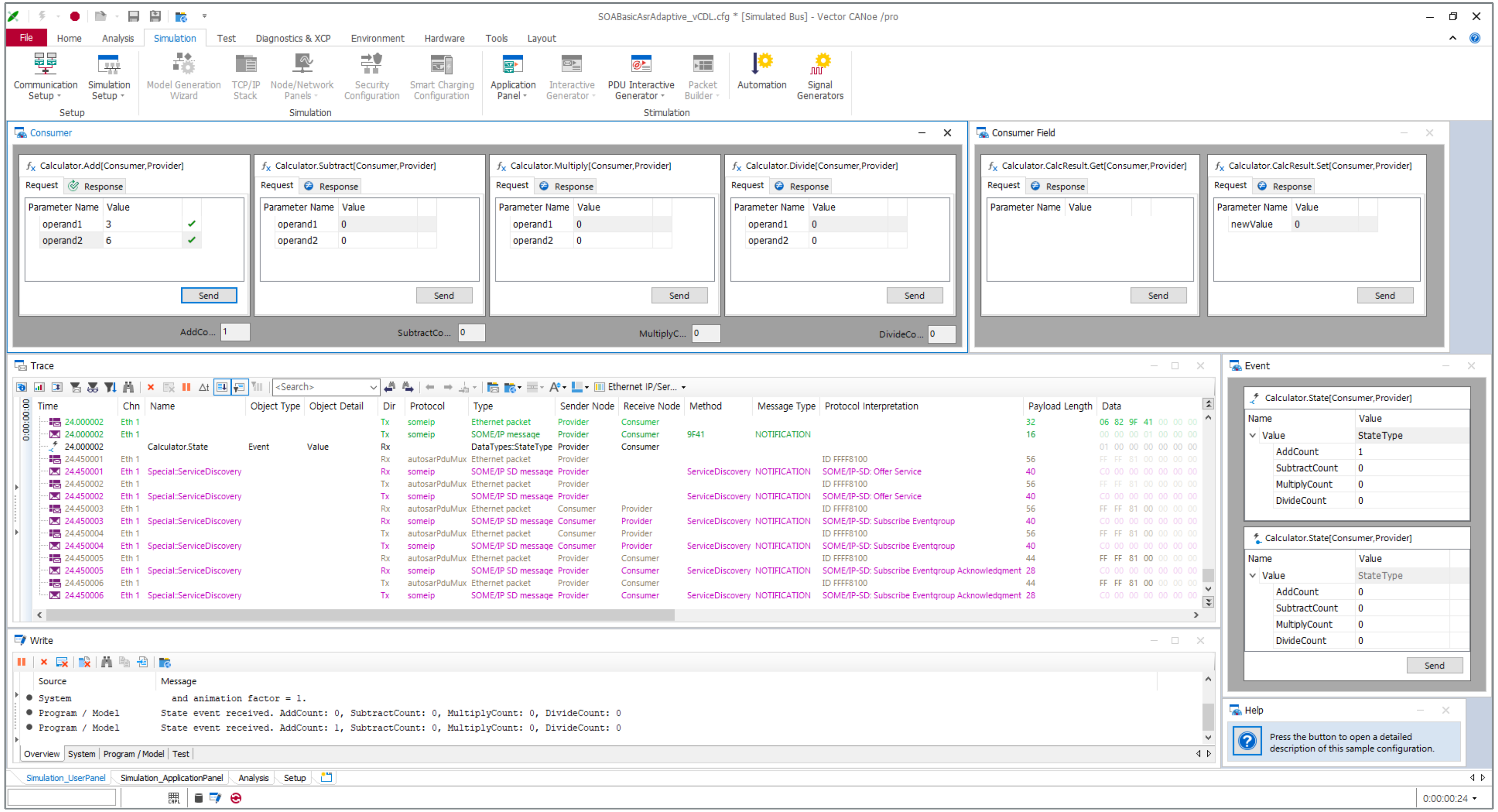
- Introduction to Ethernet- and IP-based communication
- Physical layers: IEEE 10BASE-T1S, 100BASE-T1, 1000BASE-T1, 100BASE-TX and Multi-Gig
- Overview of used communication protocols: IP, TCP, and UDP
- DoIP: Diagnostics over IP
- SOME/IP: Scalable service-Oriented MiddlewarE over IP

- CANoe for Service-Oriented Architectures(SOA) - Part1. SOA Fundamental
  - https://youtu.be/M_SXOgci1p4

VECTOR
CANoe for Service-Oriented Architectures
Part 1: Service-Orientation in E/E Designs

Calculator Example

- CANoe configuration, vCDL, Analysis Features, Simulation Features, CAPL Codes
  - Download Sample Configuration : https://portal.vector.com/shared/068f1303-77de-4eab-8bd0-ca37938c6885



Consumer

Calculator.Add[Consumer,Provider] — Request / Response

Parameter Name	Value	
operand1	3	✓
operand2	6	✓

AddCo... 1

Calculator.Subtract[Consumer,Provider] — Request / Response

Parameter Name	Value
operand1	0
operand2	0

SubtractCo... 0

Calculator.Multiply[Consumer,Provider] — Request / Response

Parameter Name	Value
operand1	0
operand2	0

MultiplyC... 0

Calculator.Divide[Consumer,Provider] — Request / Response

Parameter Name	Value
operand1	0
operand2	0

DivideCo... 0

Consumer Field

Calculator.CalcResult.Get[Consumer,Provider] — Request / Response

Parameter Name	Value

Calculator.CalcResult.Set[Consumer,Provider] — Request / Response

Parameter Name	Value
newValue	0

Trace

Time	Chn	Name	Object Type	Object Detail	Dir	Protocol	Type	Sender Node	Receive Node	Method	Message Type	Protocol Interpretation	Payload Length	Data
24.000002	Eth 1				Tx	someip	Ethernet packet	Provider	Consumer				32	06 82 9F 41 00 00 00
24.000002	Eth 1				Tx	someip	SOME/IP message	Provider	Consumer	9F41	NOTIFICATION		16	00 00 00 01 00 00 00
24.000002		Calculator.State	Event	Value	Rx		DataTypes::StateType	Provider	Consumer					01 00 00 00 00 00 00
24.450001	Eth 1				Rx	autosarPduMux	Ethernet packet	Provider				ID FFFF8100	56	FF FF 81 00 00 00 00
24.450001	Eth 1	Special::ServiceDiscovery			Rx	someip	SOME/IP SD message	Provider		ServiceDiscovery	NOTIFICATION	SOME/IP-SD: Offer Service	40	C0 00 00 00 00 00 00
24.450002	Eth 1				Tx	autosarPduMux	Ethernet packet	Provider				ID FFFF8100	56	FF FF 81 00 00 00 00
24.450002	Eth 1	Special::ServiceDiscovery			Tx	someip	SOME/IP SD message	Provider		ServiceDiscovery	NOTIFICATION	SOME/IP-SD: Offer Service	40	C0 00 00 00 00 00 00
24.450003	Eth 1				Rx	autosarPduMux	Ethernet packet	Consumer	Provider			ID FFFF8100	56	FF FF 81 00 00 00 00
24.450003	Eth 1	Special::ServiceDiscovery			Rx	someip	SOME/IP SD message	Consumer	Provider	ServiceDiscovery	NOTIFICATION	SOME/IP-SD: Subscribe Eventgroup	40	C0 00 00 00 00 00 00
24.450004	Eth 1				Tx	autosarPduMux	Ethernet packet	Consumer	Provider			ID FFFF8100	56	FF FF 81 00 00 00 00
24.450004	Eth 1	Special::ServiceDiscovery			Tx	someip	SOME/IP SD message	Consumer	Provider	ServiceDiscovery	NOTIFICATION	SOME/IP-SD: Subscribe Eventgroup	40	C0 00 00 00 00 00 00
24.450005	Eth 1				Rx	autosarPduMux	Ethernet packet	Provider	Consumer			ID FFFF8100	44	FF FF 81 00 00 00 00
24.450005	Eth 1	Special::ServiceDiscovery			Rx	someip	SOME/IP SD message	Provider	Consumer	ServiceDiscovery	NOTIFICATION	SOME/IP-SD: Subscribe Eventgroup Acknowledgment	28	C0 00 00 00 00 00 00
24.450006	Eth 1				Tx	autosarPduMux	Ethernet packet	Provider	Consumer			ID FFFF8100	44	FF FF 81 00 00 00 00
24.450006	Eth 1	Special::ServiceDiscovery			Tx	someip	SOME/IP SD message	Provider	Consumer	ServiceDiscovery	NOTIFICATION	SOME/IP-SD: Subscribe Eventgroup Acknowledgment	28	C0 00 00 00 00 00 00

Event

Calculator.State[Consumer,Provider]

Name	Value
Value	StateType
AddCount	1
SubtractCount	0
MultiplyCount	0
DivideCount	0

Calculator.State[Consumer,Provider]

Name	Value
Value	StateType
AddCount	0
SubtractCount	0
MultiplyCount	0
DivideCount	0

Write

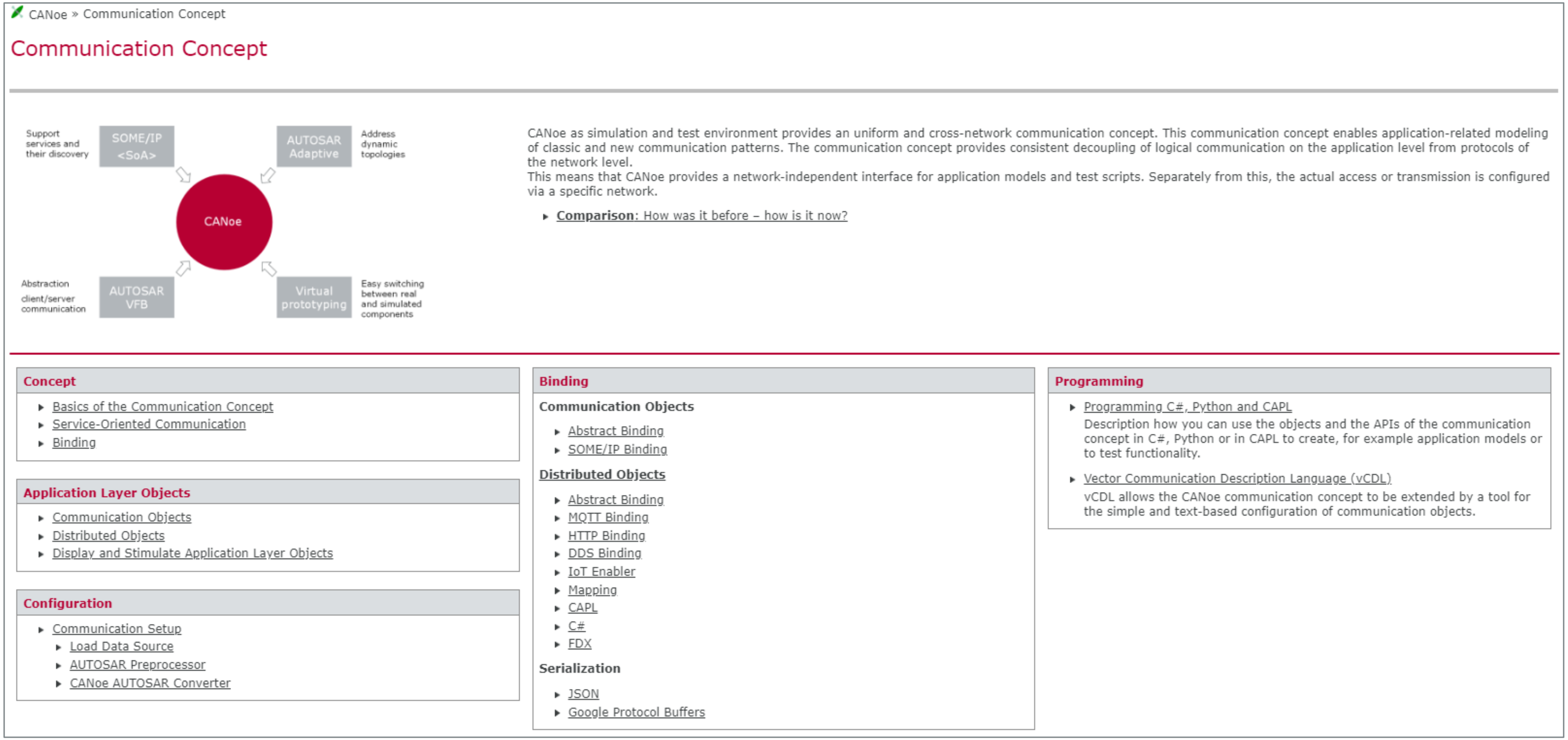
```
Source              Message
● System              and animation factor = 1.
● Program / Model    State event received. AddCount: 0, SubtractCount: 0, MultiplyCount: 0, DivideCount: 0
● Program / Model    State event received. AddCount: 1, SubtractCount: 0, MultiplyCount: 0, DivideCount: 0
```

Help

Press the button to open a detailed description of this sample configuration.

CANoe Help Manual

CANoe Communication Concept Overview


Support services and their discovery
SOME/IP <SoA>
AUTOSAR Adaptive
Address dynamic topologies
CANoe
Abstraction client/server communication
AUTOSAR VFB
Virtual prototyping
Easy switching between real and simulated components

CANoe » Communication Concept

Communication Concept

CANoe as simulation and test environment provides an uniform and cross-network communication concept. This communication concept enables application-related modeling of classic and new communication patterns. The communication concept provides consistent decoupling of logical communication on the application level from protocols of the network level.
This means that CANoe provides a network-independent interface for application models and test scripts. Separately from this, the actual access or transmission is configured via a specific network.

- **Comparison**: How was it before – how is it now?

Concept

- Basics of the Communication Concept
- Service-Oriented Communication
- Binding

Application Layer Objects

- Communication Objects
- Distributed Objects
- Display and Stimulate Application Layer Objects

Configuration

- Communication Setup
  - Load Data Source
  - AUTOSAR Preprocessor
  - CANoe AUTOSAR Converter

Binding

Communication Objects

- Abstract Binding
- SOME/IP Binding

Distributed Objects

- Abstract Binding
- MQTT Binding
- HTTP Binding
- DDS Binding
- IoT Enabler
- Mapping
- CAPL
- C#
- FDX

Serialization

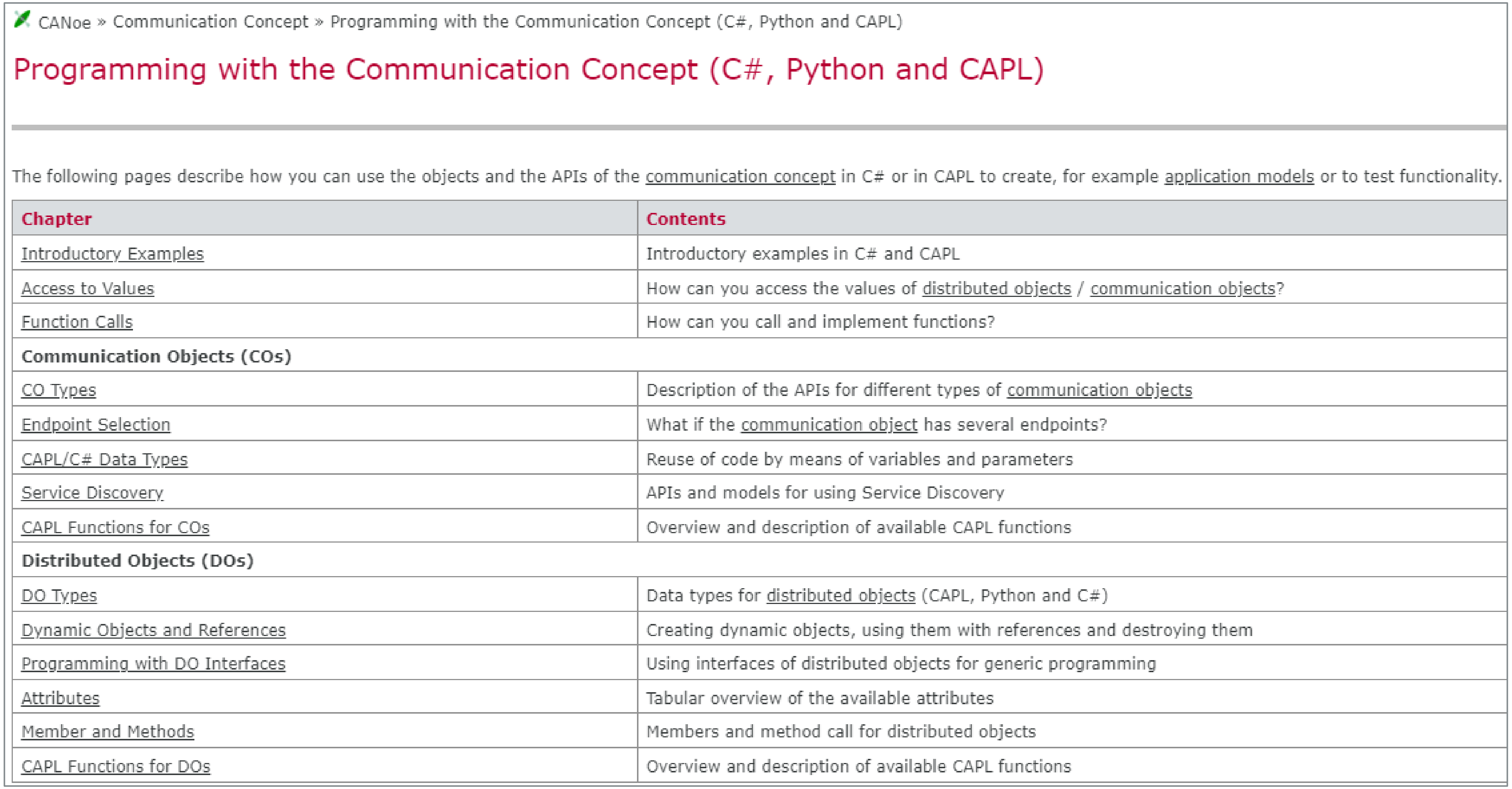
- JSON
- Google Protocol Buffers

Programming

- Programming C#, Python and CAPL
  Description how you can use the objects and the APIs of the communication concept in C#, Python or in CAPL to create, for example application models or to test functionality.
- Vector Communication Description Language (vCDL)
  vCDL allows the CANoe communication concept to be extended by a tool for the simple and text-based configuration of communication objects.

CANoe Help Manual

CAPL Programming Overview



CANoe » Communication Concept » Programming with the Communication Concept (C#, Python and CAPL)

Programming with the Communication Concept (C#, Python and CAPL)

The following pages describe how you can use the objects and the APIs of the communication concept in C# or in CAPL to create, for example application models or to test functionality.

Chapter	Contents
Introductory Examples	Introductory examples in C# and CAPL
Access to Values	How can you access the values of distributed objects / communication objects?
Function Calls	How can you call and implement functions?
Communication Objects (COs)	
CO Types	Description of the APIs for different types of communication objects
Endpoint Selection	What if the communication object has several endpoints?
CAPL/C# Data Types	Reuse of code by means of variables and parameters
Service Discovery	APIs and models for using Service Discovery
CAPL Functions for COs	Overview and description of available CAPL functions
Distributed Objects (DOs)	
DO Types	Data types for distributed objects (CAPL, Python and C#)
Dynamic Objects and References	Creating dynamic objects, using them with references and destroying them
Programming with DO Interfaces	Using interfaces of distributed objects for generic programming
Attributes	Tabular overview of the available attributes
Member and Methods	Members and method call for distributed objects
CAPL Functions for DOs	Overview and description of available CAPL functions

For more information about Vector
and our products please visit

www.vector.com

Author:
Lee, Jaecheol
Vector Korea