Tutorial

CIP Statemachine - Lamp

Tutorial	Actifsource Tutorial – CIP Statemachine - Lamp
Required Time	• 60 Minutes
Prerequisites	• Actifsource Tutorial – Installing Actifsource • Actifsource Tutorial – Simple Service
Goal	• Creating a state machine using the CIP method • Generating real time C code for any embedded system
Topics covered	• Setting up a new CIP Project • Communicating with the Outer World • Specify the State Machine • Generating State Machine Code
Notation	↳ To do ⓘ Information • **Bold**: Terms from actifsource or other technologies and tools • <u>**Bold underlined**</u>: actifsource Resources • <u>Underlined</u>: User Resources • <u>*UnderlinedItalics*</u>: Resource Functions • `Monospaced`: User input • *Italics*: Important terms in current situation
Disclaimer	The authors do not accept any liability arising out of the application or use of any information or equipment described herein. The information contained within this document is by its very nature incomplete. Therefore the authors accept no responsibility for the precise accuracy of the documentation contained herein. It should be used rather as a guide and starting point.
Contact	**actifsource GmbH** Täfernstrasse 37 5405 Baden-Dättwil Switzerland <u>www.actifsource.com</u>
Trademark	**actifsource** is a registered trademark of **actifsource GmbH** in Switzerland, the EU, USA, and China. Other names appearing on the site may be trademarks of their respective owners.
Compatibility	Created with **actifsource** Version 5.8.7

Overview

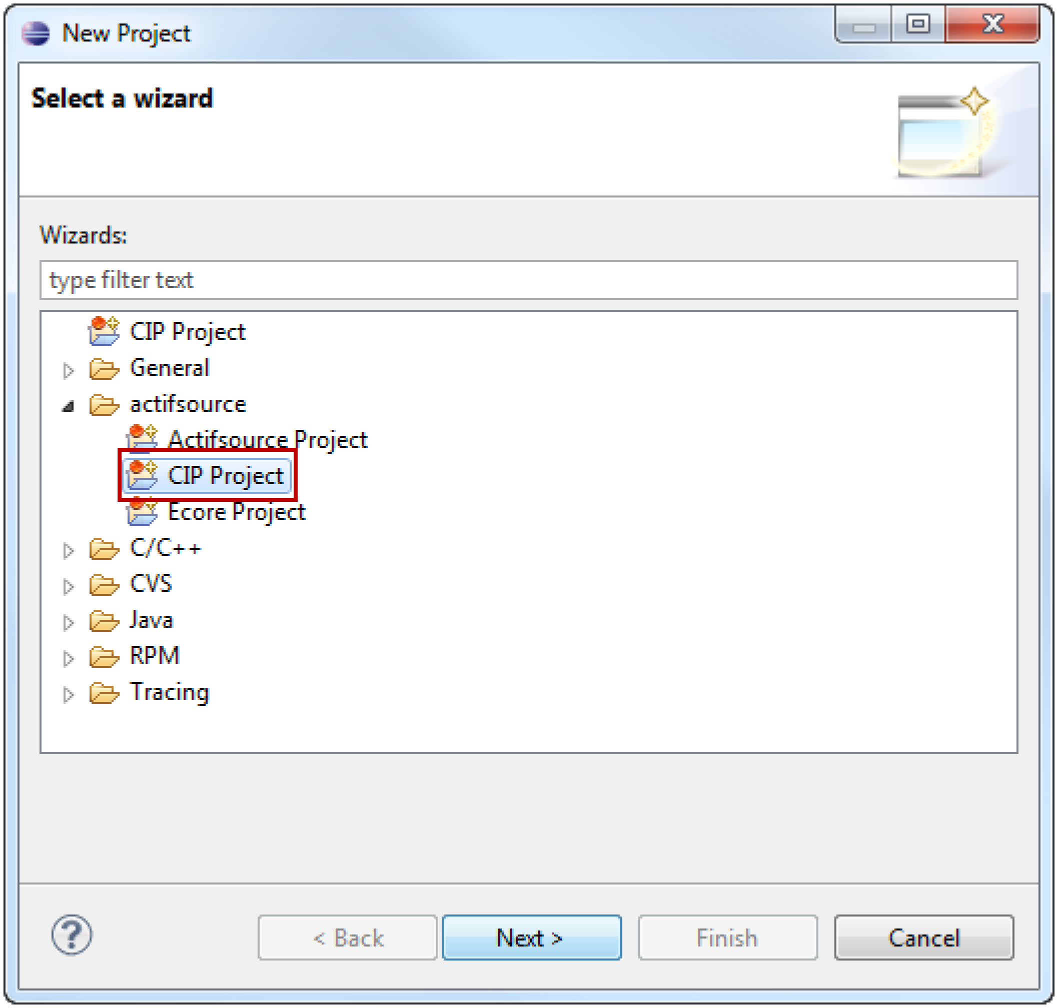
- Learn how to specify a simple state machine
- Example
    - Button to turn on and off a lamp
    - Turning off the lamp shall be delayed
- CIP Method
    - The CIP System is the root element
    - The CIP System consists of Clusters
        - Clusters are used to model distributed state machines
    - The CIP Cluster consists of Processes
        - The process declares the state of the state machine
    - The CIP Process consists of Modes
        - Modes are used for different situations like normal, error, run-in, run-out
        - The mode declares the transitions between the states

Part I:

Setting up a new CIP Project

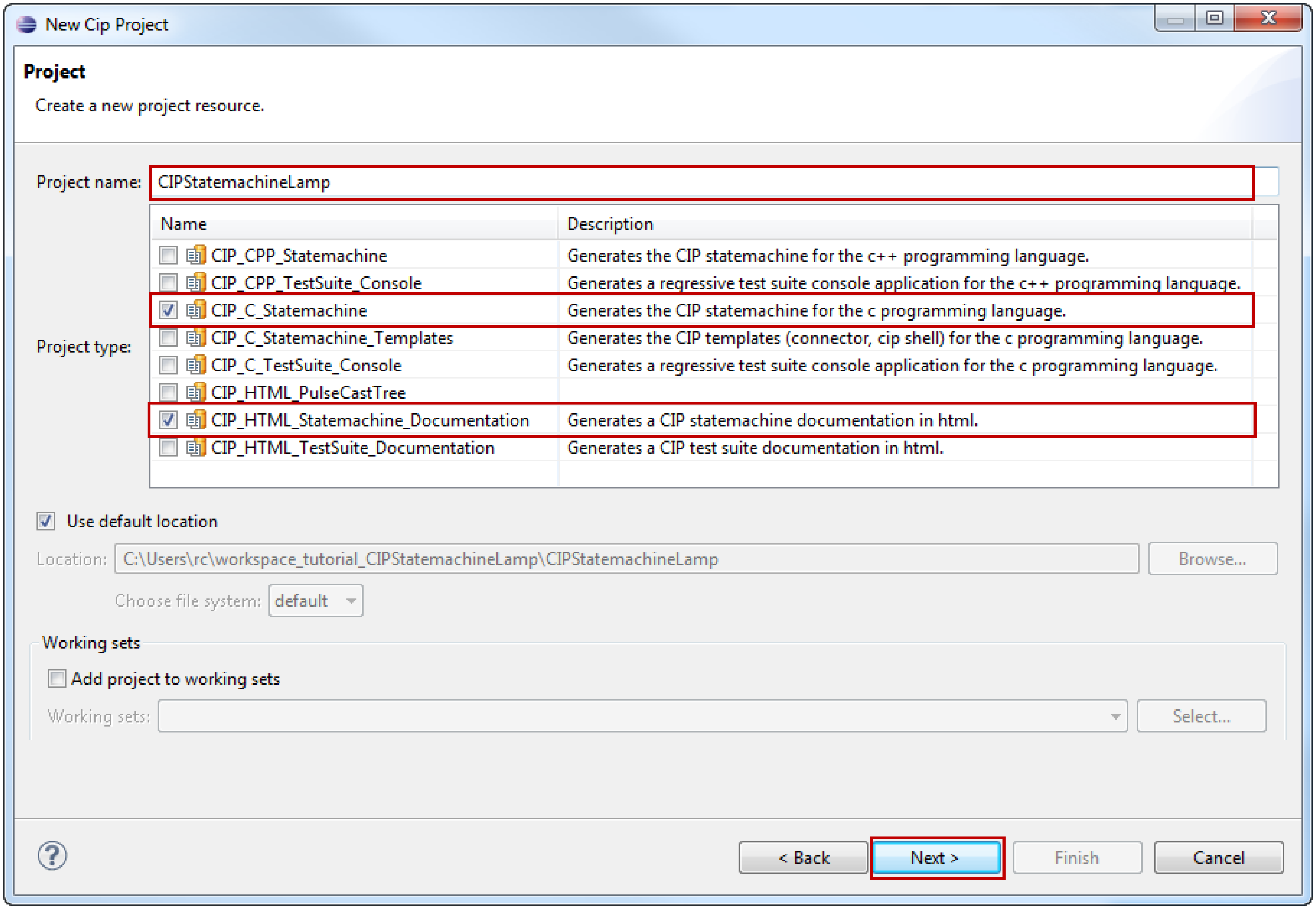
- Create a new CIP Project with the CIP Project wizard

Setting up a new CIP Project



- Prepare a new **Actifsource/CIP Project** using the Actifsource CIP wizard
  - File/new/other
  - Actifsource/CIP Project
- Click *Next*

Setting up a new CIP Project



Project

Create a new project resource.

Project name: CIPStatemachineLamp

Project type:

	Name	Description
☐	CIP_CPP_Statemachine	Generates the CIP statemachine for the c++ programming language.
☐	CIP_CPP_TestSuite_Console	Generates a regressive test suite console application for the c++ programming language.
☑	CIP_C_Statemachine	Generates the CIP statemachine for the c programming language.
☐	CIP_C_Statemachine_Templates	Generates the CIP templates (connector, cip shell) for the c programming language.
☐	CIP_C_TestSuite_Console	Generates a regressive test suite console application for the c programming language.
☐	CIP_HTML_PulseCastTree	
☑	CIP_HTML_Statemachine_Documentation	Generates a CIP statemachine documentation in html.
☐	CIP_HTML_TestSuite_Documentation	Generates a CIP test suite documentation in html.

☑ Use default location

Location: C:\Users\rc\workspace_tutorial_CIPStatemachineLamp\CIPStatemachineLamp

Choose file system: default

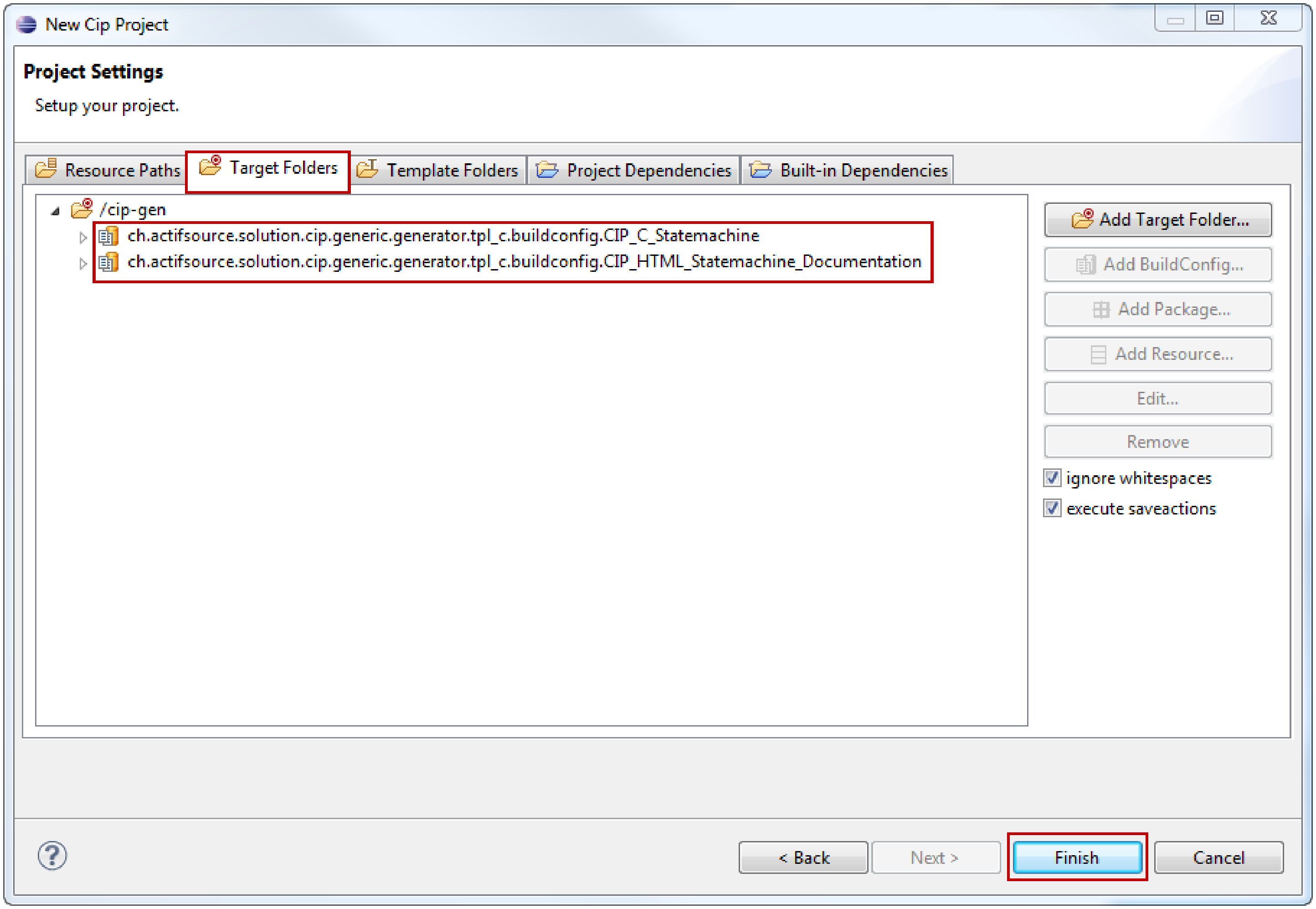
Working sets

☐ Add project to working sets

Working sets:

- Specifiy *Project name*
- Specify *Project type*
- Click *Next*

Setting up a new CIP Project



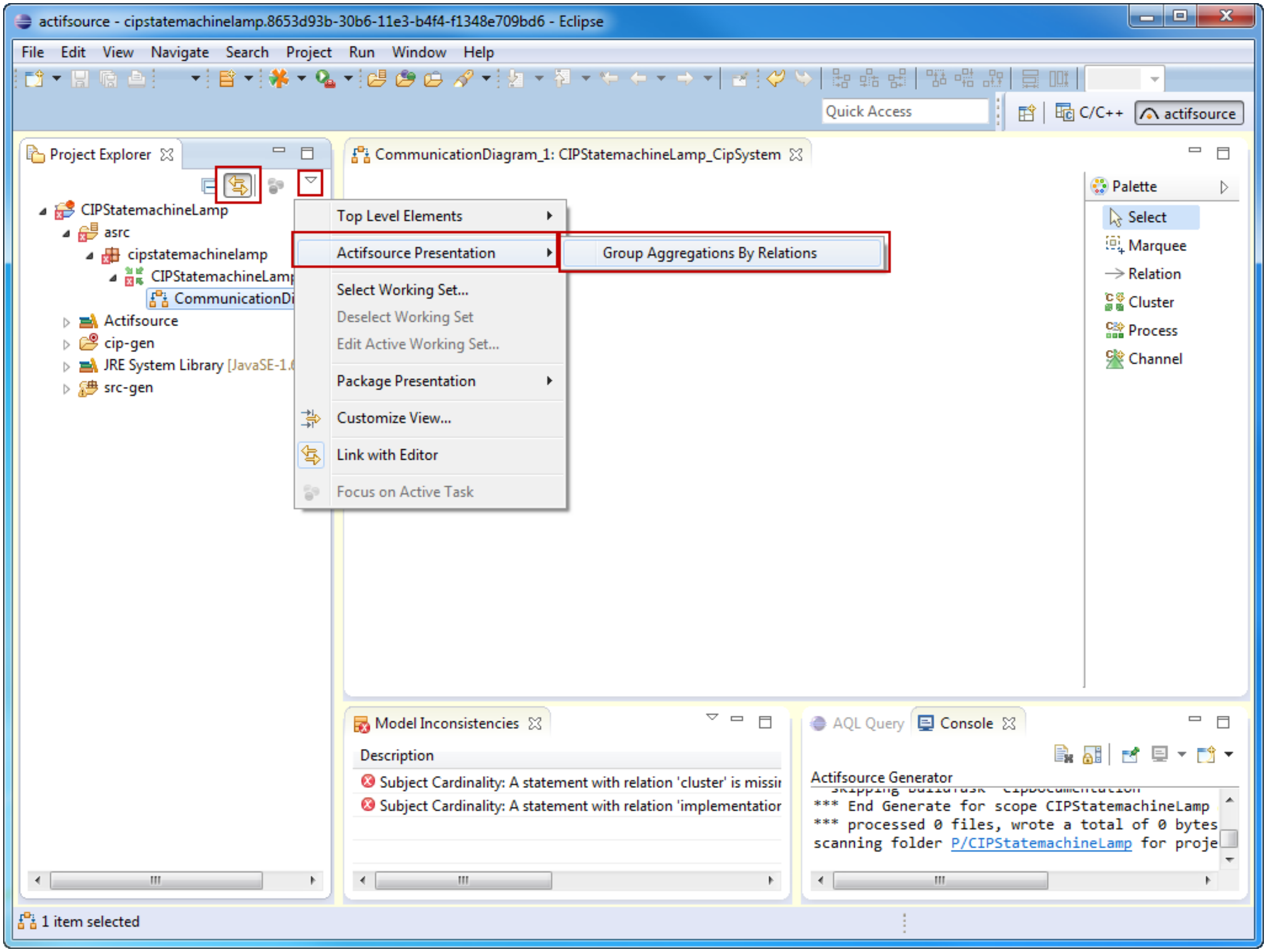
/cip-gen
- ch.actifsource.solution.cip.generic.generator.tpl_c.buildconfig.CIP_C_Statemachine
- ch.actifsource.solution.cip.generic.generator.tpl_c.buildconfig.CIP_HTML_Statemachine_Documentation

[✓] ignore whitespaces
[✓] execute saveactions

ⓘ Note that the BuildConfigs in TargetFolder are equivalent to the previously selected project type

ⓘ You may add other build configs (i.e. test suites) as needed

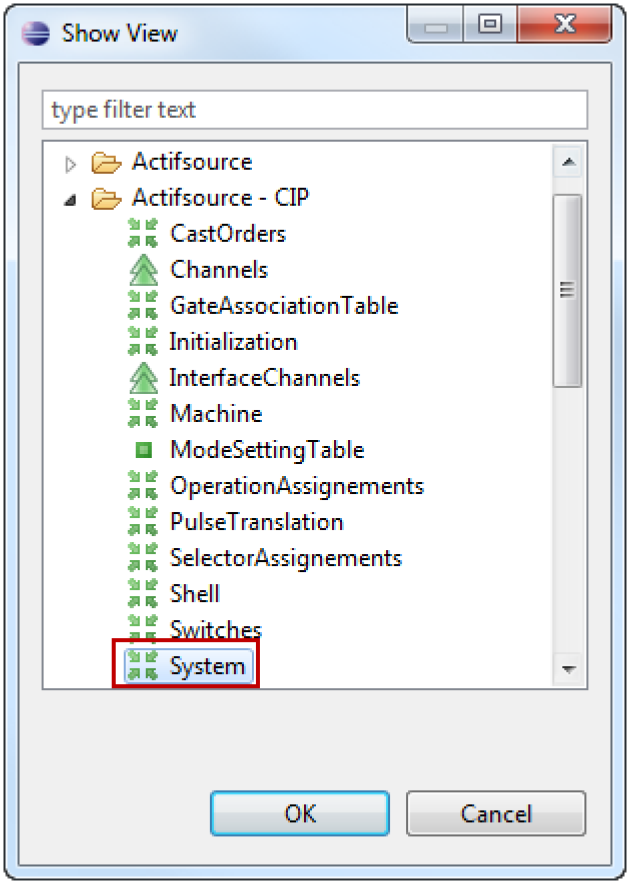
↳ Click *Finish*



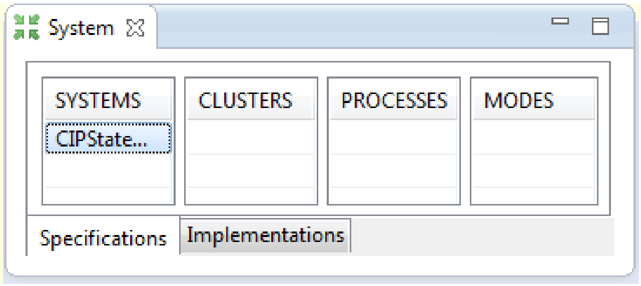
Subject Cardinality: A statement with relation 'cluster' is missir
Subject Cardinality: A statement with relation 'implementatior

```
*** End Generate for scope CIPStatemachineLamp
*** processed 0 files, wrote a total of 0 bytes
scanning folder P/CIPStatemachineLamp for proje
```

- *Link with Editor*
- Enable *Actifsource Presentation* Flag Group Aggregations By Relations

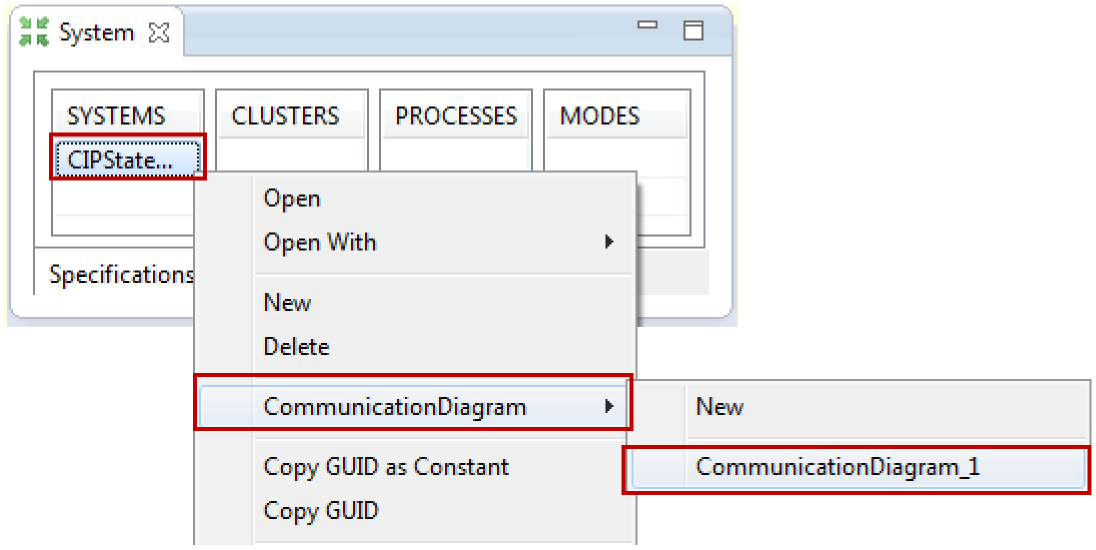


- Open *CIP System View*
    - Window/Show View/Other…
    - Actifsource – CIP / System



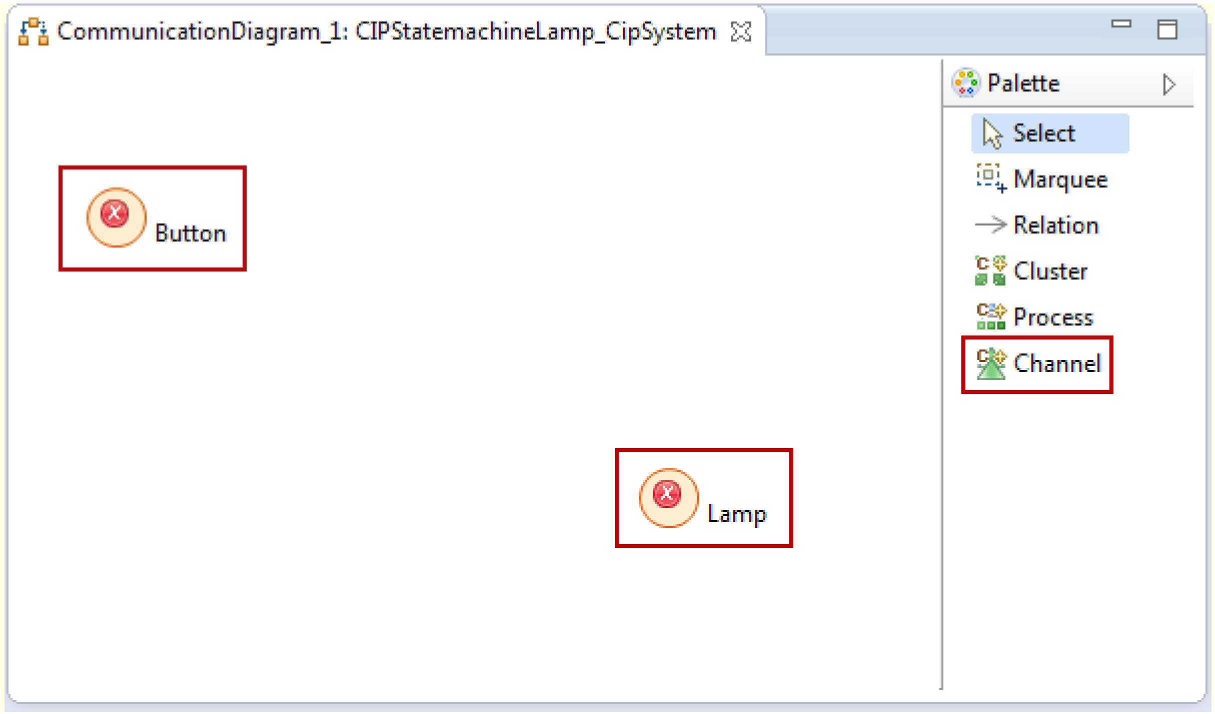
Part II: Communicating with the Outer World

- Let’s communicate with the outer world
- Channels are providing messages from physical device

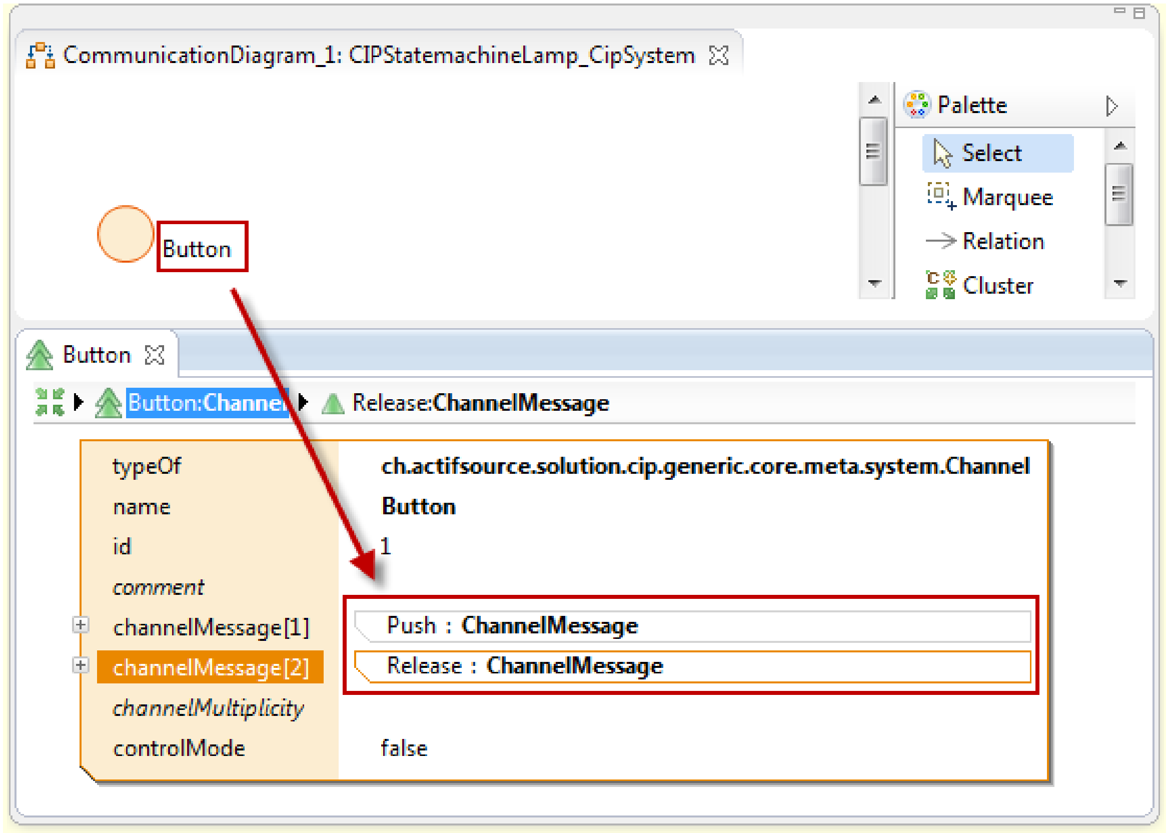


- Open the Communication Diagram (if not already open)
  - Right Click on System1 in the *System View*
  - Select *CommunicationDiagram* from the context menu
  - Open CommunicationDiagram_1

Communicating with the Outer World



↳ Create two new **Channels** named Button and Lamp using the *Channel* tool from the *Palette*

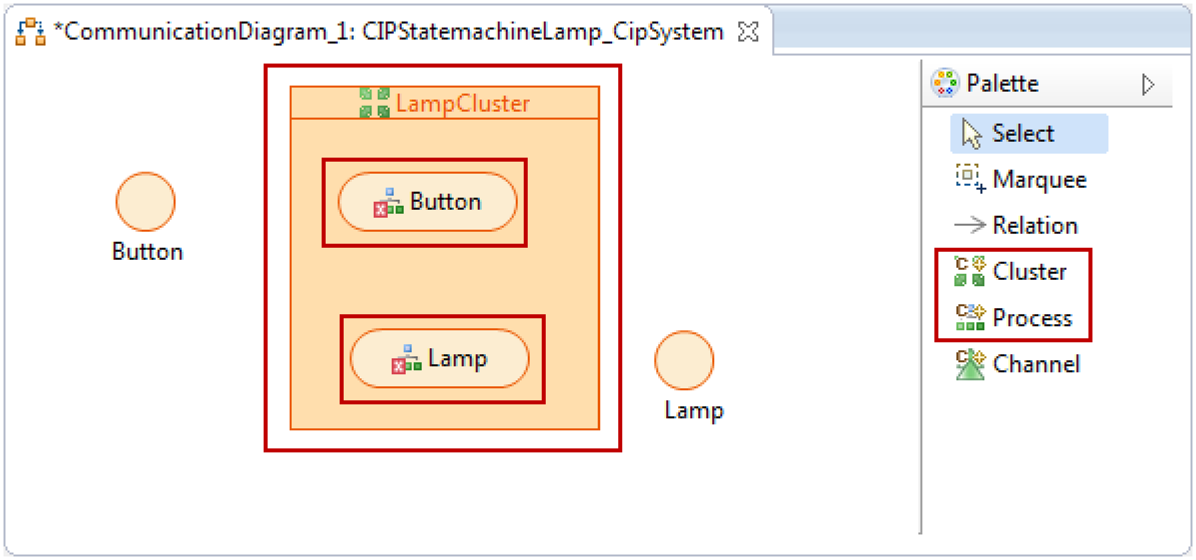


typeOf	ch.actifsource.solution.cip.generic.core.meta.system.Channel
name	Button
id	1
comment	
channelMessage[1]	Push : ChannelMessage
channelMessage[2]	Release : ChannelMessage
channelMultiplicity	
controlMode	false

ⓘ Arrange the Graphical Editor and the Resource Editor together on the same screen as shown above

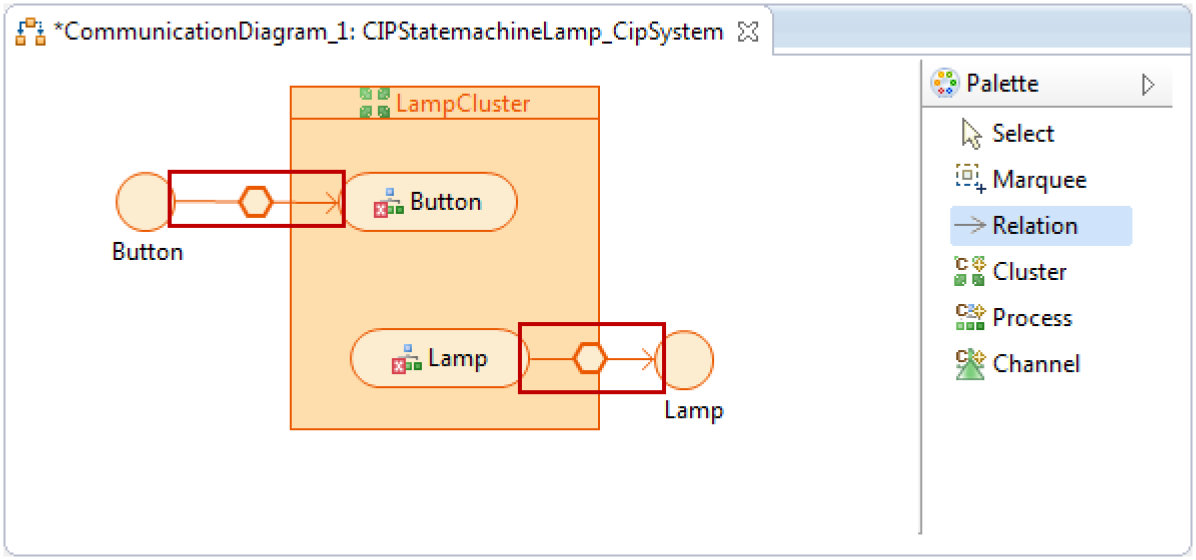
- Add the two **ChannelMessages** Push and Release to the **Channel** Button
  - Ctrl+Click on the Button label
  - Add **ChannelMessage** Push and **ChannelMessage** Release
- Add the two **ChannelMessages** Bright and Dark to the **Channel** Lamp
  - Ctrl+Click on the Lamp label
  - Add **ChannelMessage** Bright and **ChannelMessage** Dark

ⓘ Messages are given as function calls from the other world to the state machine and vice versa

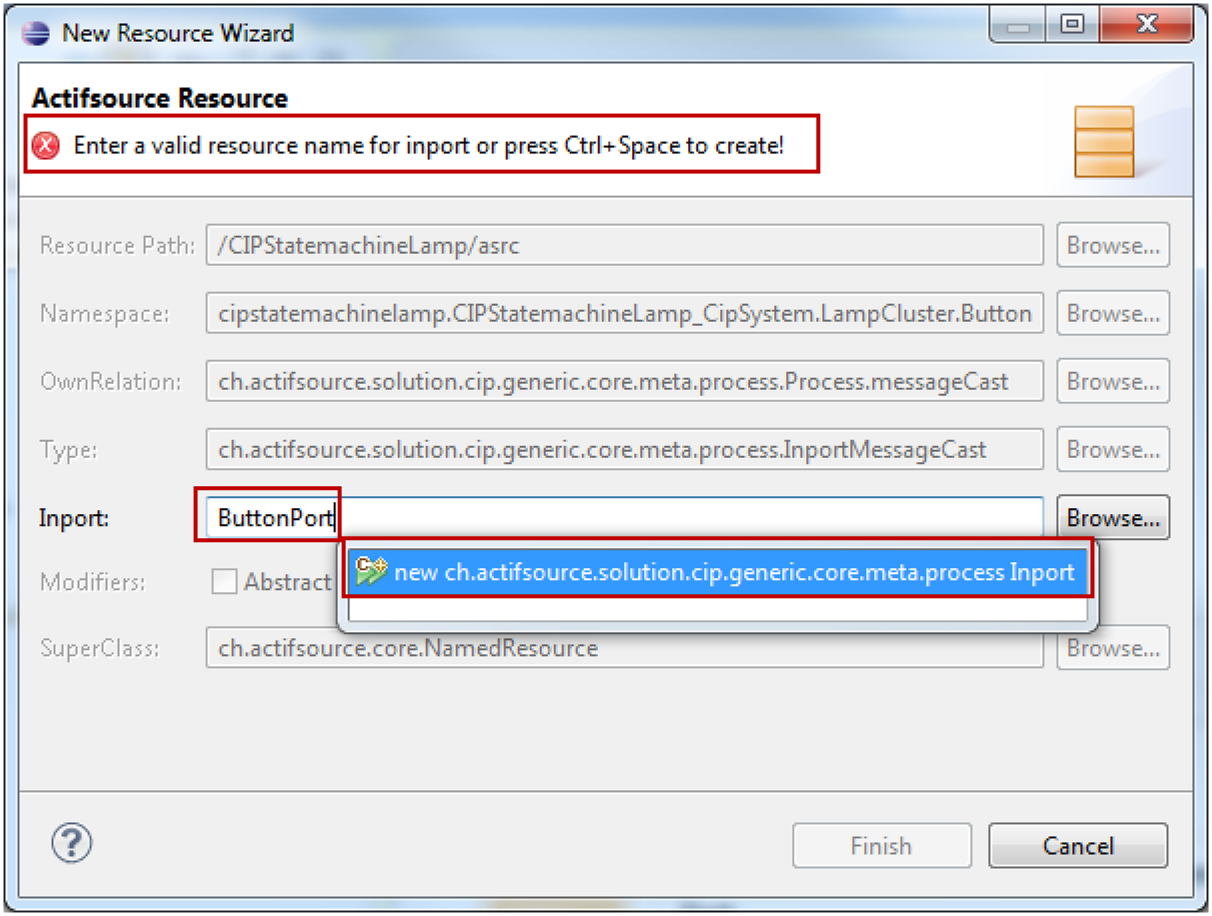


- ⓘ The CIP Method specifies that every physical process needs a logical counterpart in the model
- ⓘ Note that different Clusters may run on different processors. This allows you to design distributed state machines using the CIP Method.
- ↳ Add the two **Processes** Button and Lamp to the **Cluster** LampCluster

Communicating with the Outer World



- ⓘ Channels are Delivering Messages to a Process via Port. Doing so allows you to consider the Process to a self-consistent component.
- Create an input relation from the **Channel** Button to the **Process** Button via the **Port** ButtonPort
- Create an output relation from the **Process** Lamp to the **Channel** Lamp via the **Port** LampPort



Actifsource Resource

Enter a valid resource name for inport or press Ctrl+Space to create!

Resource Path: /CIPStatemachineLamp/asrc

Namespace: cipstatemachinelamp.CIPStatemachineLamp_CipSystem.LampCluster.Button

OwnRelation: ch.actifsource.solution.cip.generic.core.meta.process.Process.messageCast

Type: ch.actifsource.solution.cip.generic.core.meta.process.InportMessageCast

Inport: ButtonPort

new ch.actifsource.solution.cip.generic.core.meta.process Inport

Modifiers: Abstract

SuperClass: ch.actifsource.core.NamedResource

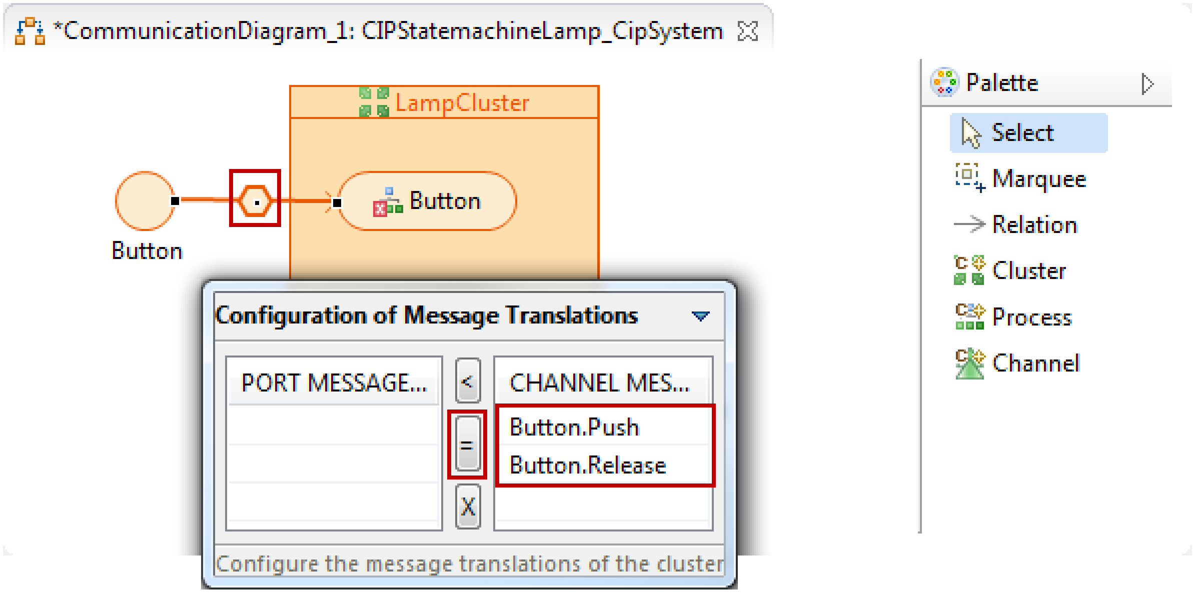
ⓘ If you create a relation from a channel to a process you have to specify a port

↳ Enter the desired **Port** name ButtonPort and press Ctrl-Space

- Select an existing **Port**

  or
- Create a new **Port** (as we do in our example)

Communicating with the Outer World


*CommunicationDiagram_1: CIPStatemachineLamp_CipSystem
LampCluster
Button
Button
Palette
Select
Marquee
Relation
Cluster
Process
Channel
Configuration of Message Translations
PORT MESSAGE...
CHANNEL MES...
Button.Push
Button.Release
<
=
X
Configure the message translations of the cluster

ⓘ Since every **Process** is a self-consistent component we have to specify a **Message** interface on the port to. The **=** tool in the *Message Translation* helps us to create the same messages as found on the **Channel** also on the corresponding **Port**.

↳ Configure the *Message Translation* between the **Channel** Button and the **Process** Button
- Double-Click on the hexagon of the relation between the **Channel** Button and the **Process** Button
- Select the **ChannelMessage**s Button.Push and Button.Release
- Use the =-Tool to create the corresponding **Messages** on the **Port** ButtonPort of the **Process** Button

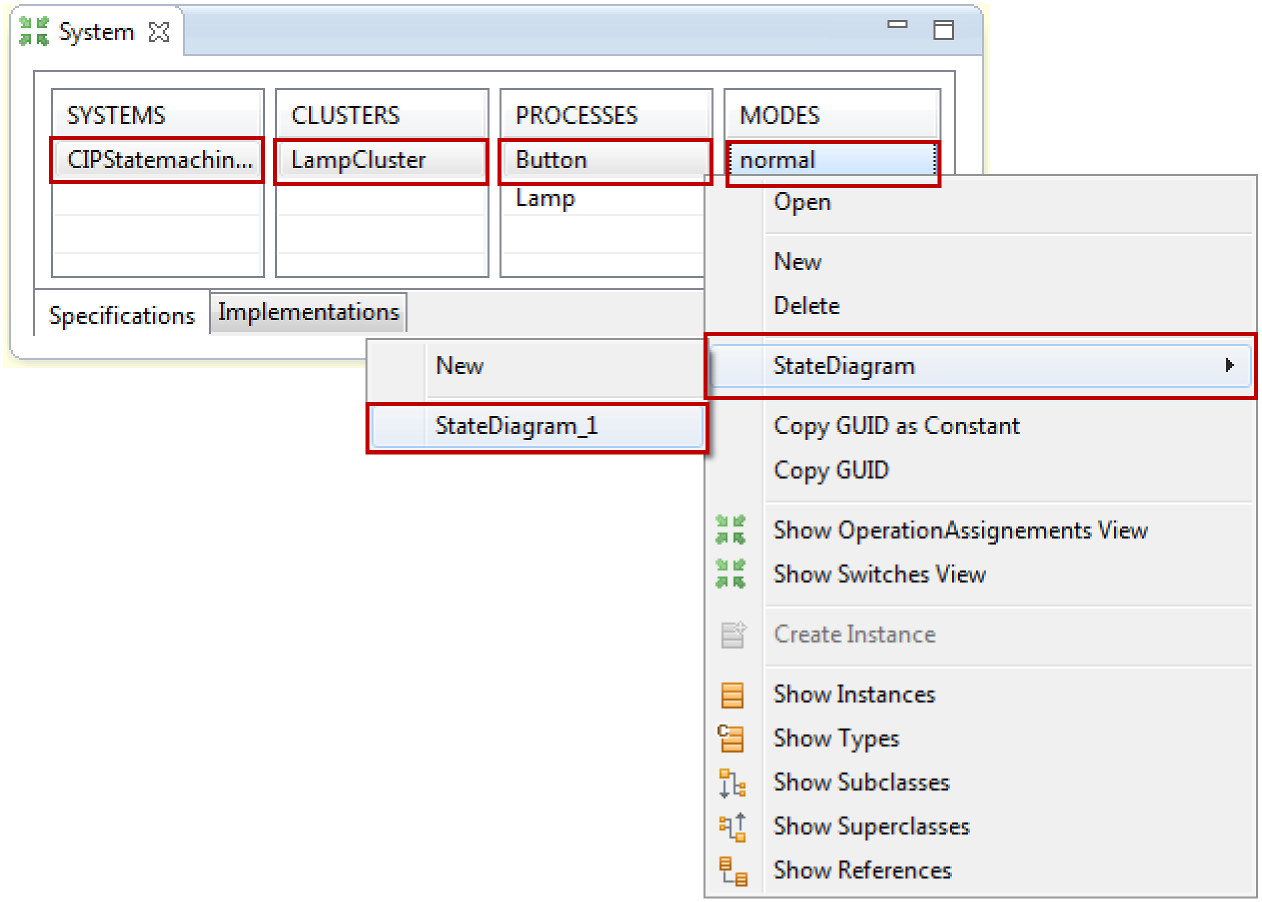
↳ Configure the *Message Translation* between the **Process** Lamp and the **Channel** Lamp
- Double-Click on the hexagon of the relation between the the **Process** Lamp and the **Channel** Lamp
- Select the **ChannelMessage**s Lamp.Bright and Lamp.Dark
- Use the =-Tool to create the corresponding **Messages** on the **Port** LampPort of the **Process** Lamp

Part III:

Specify the State Machine

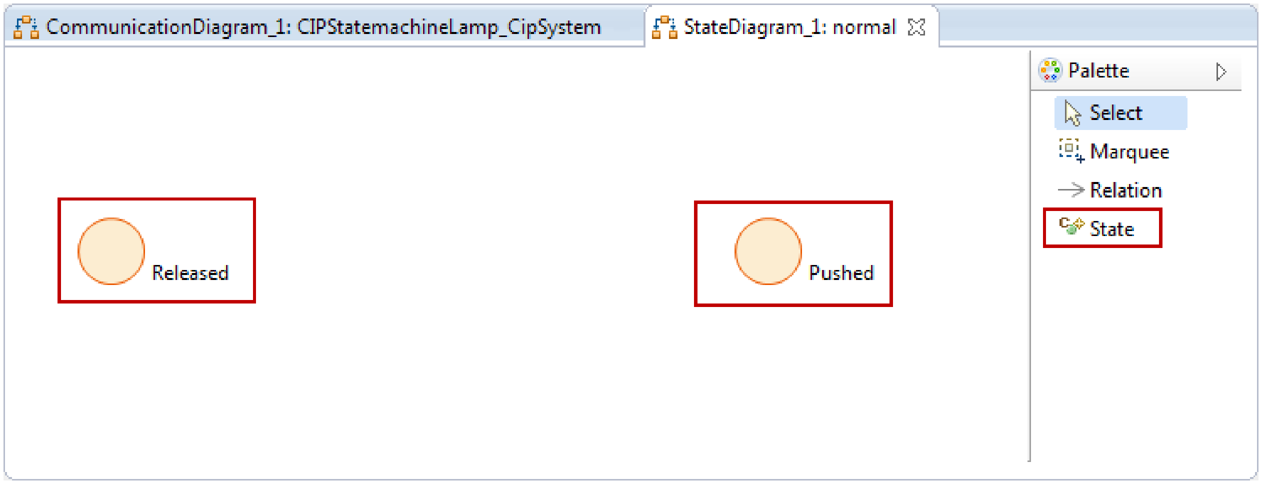
- We are now set to specify the state machines for each process
- Note that the CIP Method knows so called **Modes**
  - A Process can have one or more Modes
  - States are declared by the Process
  - Transitions are declared by the Mode
- Making this difference it becomes possible to provide several Modes for several situations
  - Normal Mode
  - Error Mode
  - Run-In Mode
  - Run-Out Mode
- There are the following rules
  - **States** are defined in the **Process**
  - **States** are shared for every **Mode**
  - **Transitions** are defined in the **Mode**

Specify the State Machine



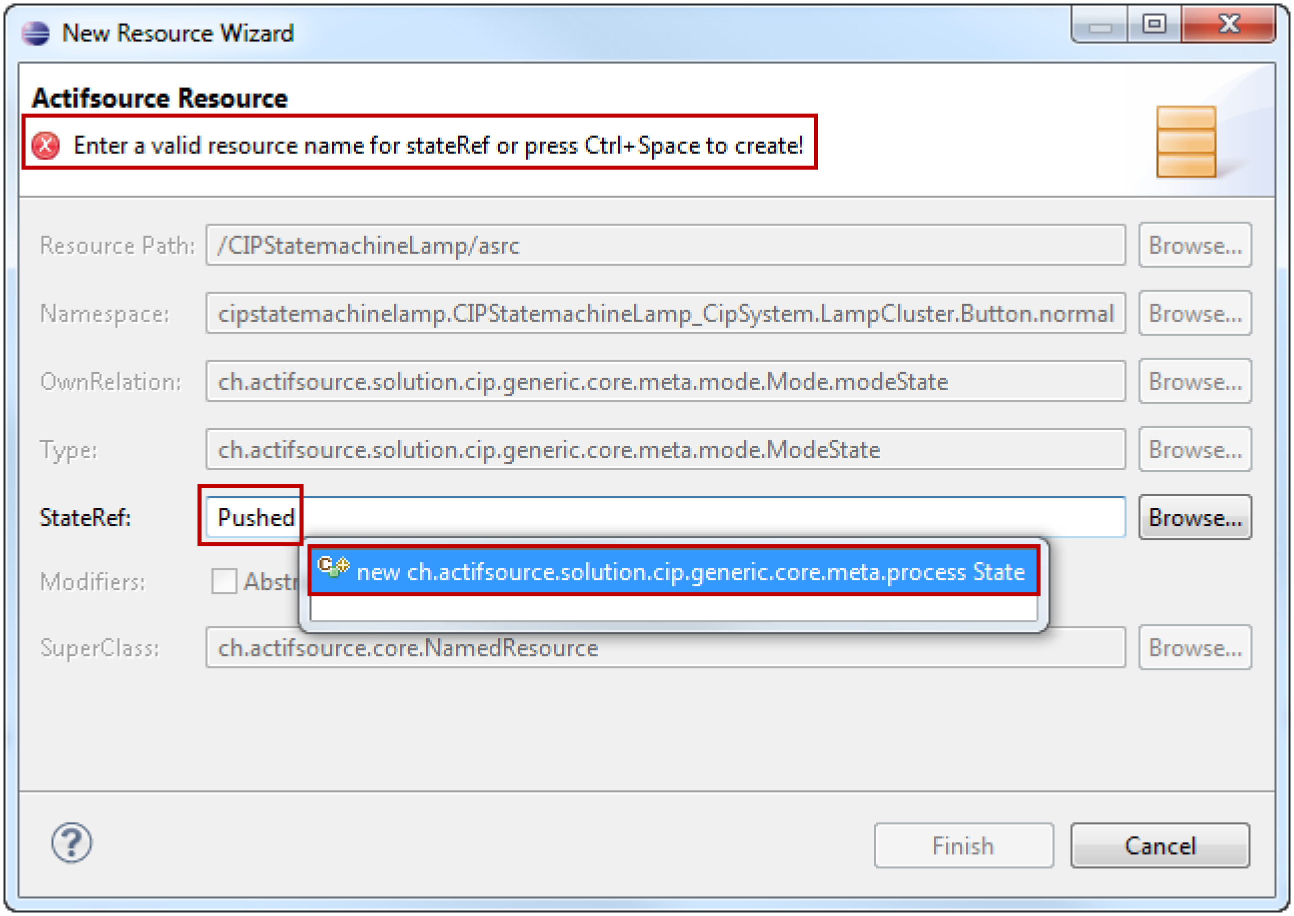
- Open the *StateDiagram* for **Process** Button
  - On the *System View* Right-Click on System1.Lamp.Button.normal
  - Select *StateDiagram*
  - Select StateDiagram_1

Specify the State Machine



↳ Enter two new **States** Released and Pushed using the *Palette*

Specify the State Machine


New Resource Wizard

Actifsource Resource

Enter a valid resource name for stateRef or press Ctrl+Space to create!

Resource Path: /CIPStatemachineLamp/asrc

Namespace: cipstatemachinelamp.CIPStatemachineLamp_CipSystem.LampCluster.Button.normal

OwnRelation: ch.actifsource.solution.cip.generic.core.meta.mode.Mode.modeState

Type: ch.actifsource.solution.cip.generic.core.meta.mode.ModeState

StateRef: Pushed

new ch.actifsource.solution.cip.generic.core.meta.process State

Modifiers: Abstr

SuperClass: ch.actifsource.core.NamedResource

ⓘ State are shared for every mode

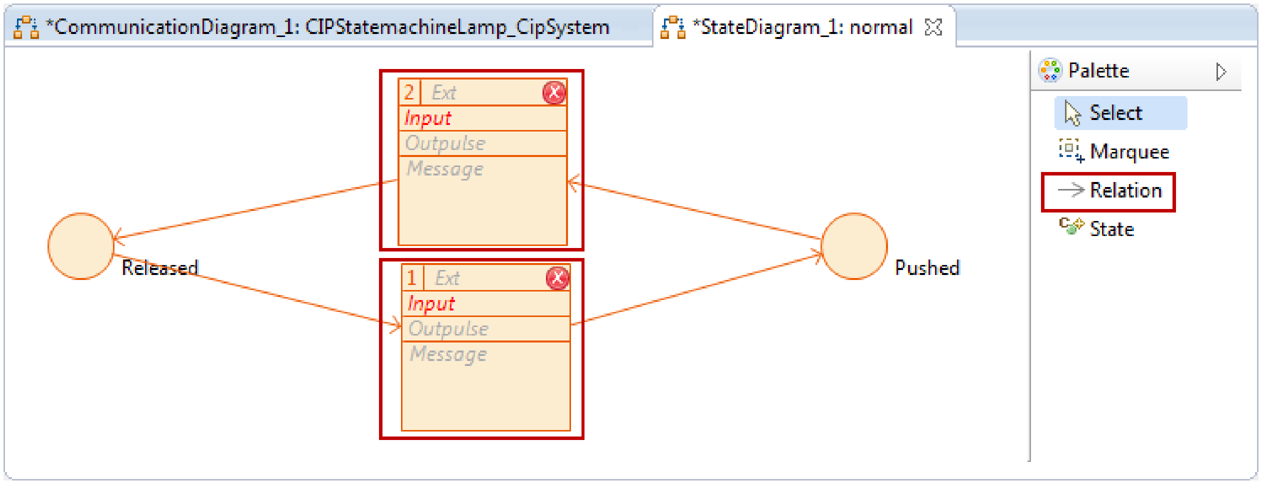
↳ Enter the desired **State** name Released and press Ctrl-Space

- Select an existing **State**

  or

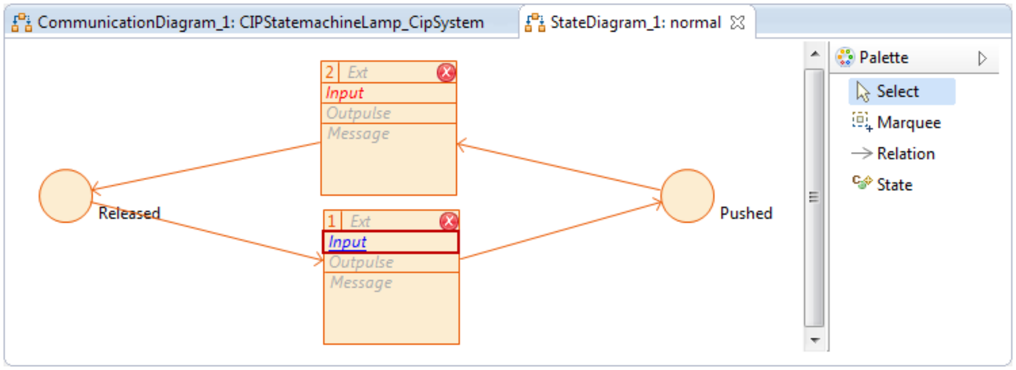
- Create a new **State** (as we do in our example)

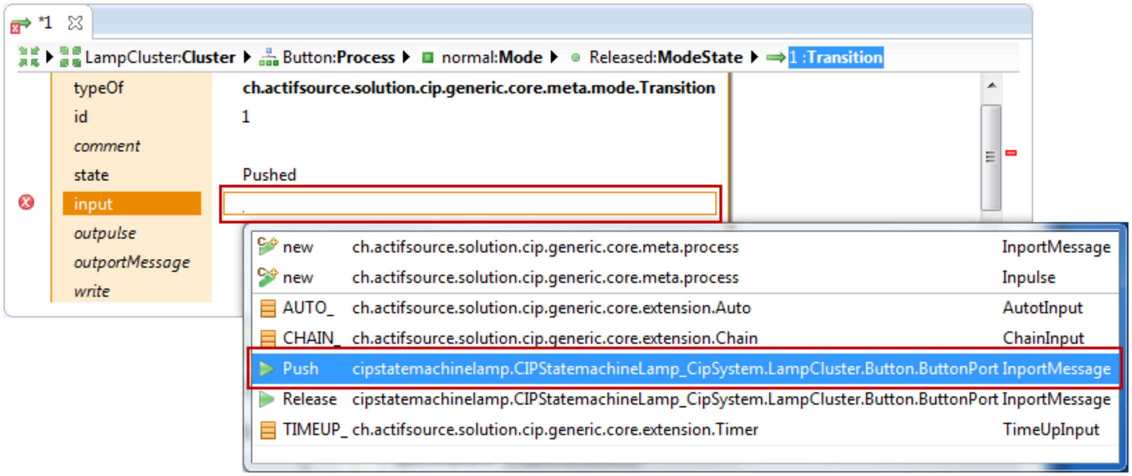
Specify the State Machine



- Create a new **Transition** from **State** Released to **State** Pushed using the *Relation* Tool from the *Palette*
- Create a new **Transition** from **State** Pushed to **State** Released using the *Relation* Tool from the *Palette*

Specify the State Machine




*1
LampCluster:Cluster ▸ Button:Process ▸ normal:Mode ▸ Released:ModeState ▸ 1 :Transition

typeOf	ch.actifsource.solution.cip.generic.core.meta.mode.Transition
id	1
comment	
state	Pushed
input	
outpulse	
outportMessage	
write	

new	ch.actifsource.solution.cip.generic.core.meta.process	InportMessage
new	ch.actifsource.solution.cip.generic.core.meta.process	Inpulse
AUTO_	ch.actifsource.solution.cip.generic.core.extension.Auto	AutotInput
CHAIN_	ch.actifsource.solution.cip.generic.core.extension.Chain	ChainInput
Push	cipstatemachinelamp.CIPStatemachineLamp_CipSystem.LampCluster.Button.ButtonPort	InportMessage
Release	cipstatemachinelamp.CIPStatemachineLamp_CipSystem.LampCluster.Button.ButtonPort	InportMessage
TIMEUP_	ch.actifsource.solution.cip.generic.core.extension.Timer	TimeUpInput

ⓘ Every Transition needs an input to be triggered

ⓘ Note that the Actifsource Validator marks all resources that are incomplete

↳ Add the input message to **Transition** from **State** Released to **State** Pushed

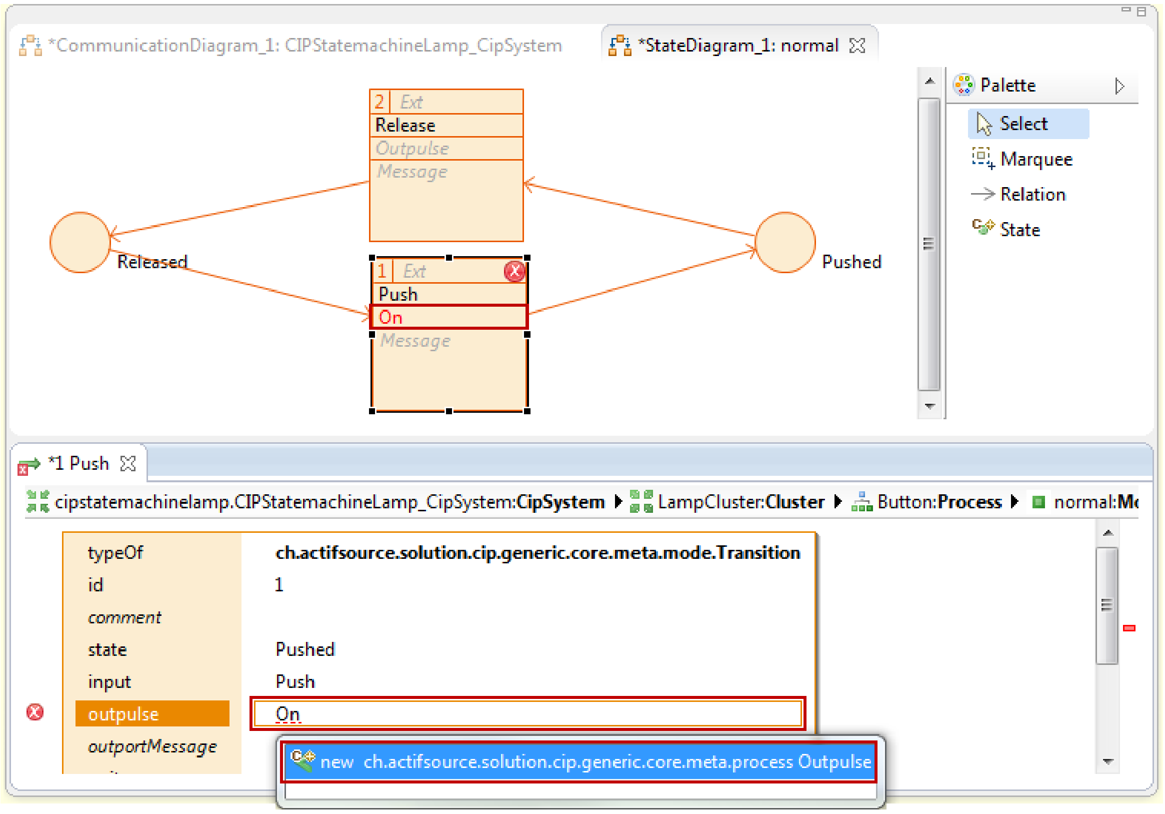
- Ctrl-Klick on the italics label *Input*
- Select the **InportMessage** Push

↳ Add the input message to **Transition** from **State** Pushed to **State** Released

- Ctrl-Klick on the italics label *Input*
- Select the **InportMessage** Release

ⓘ Note that you now selecting the **Messages** that we have previously created on the **Port** Button using the =-Tool in the *Message Translation*

Specify the State Machine



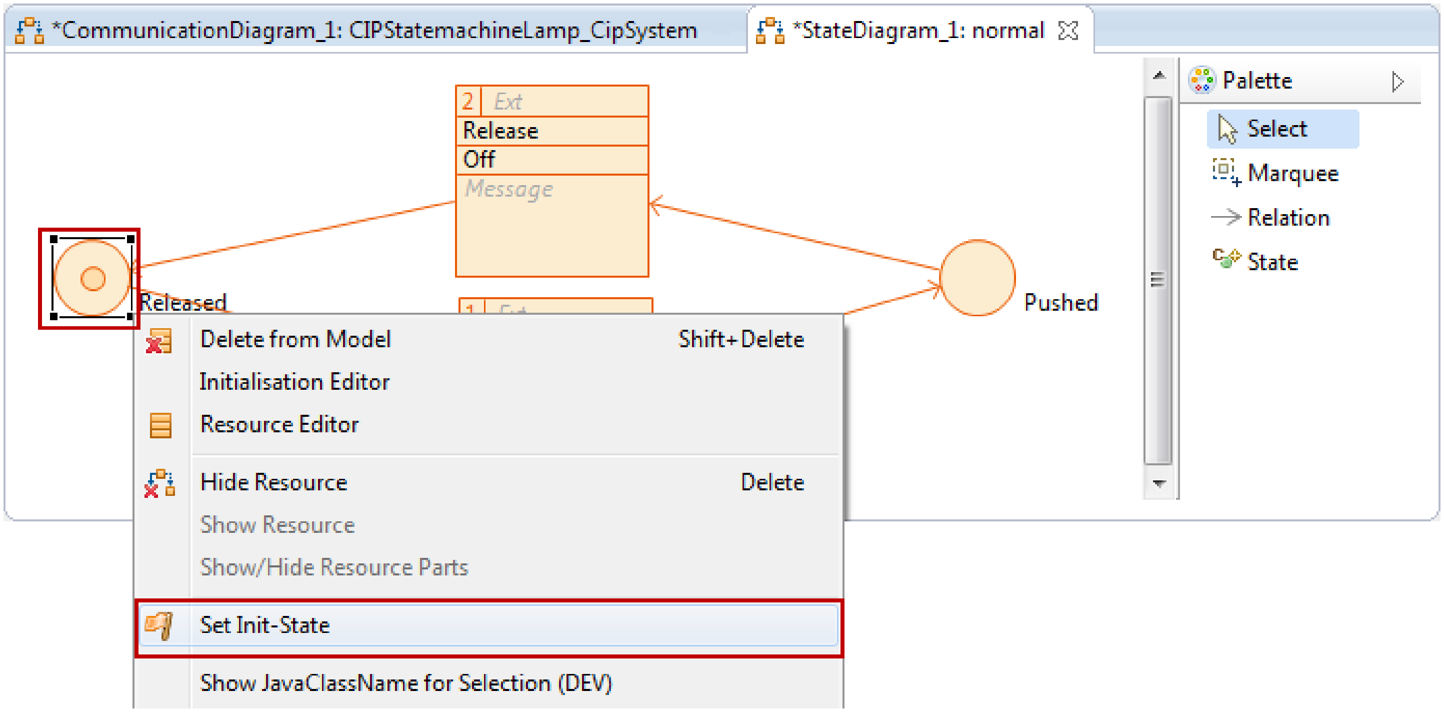
cipstatemachinelamp.CIPStatemachineLamp_CipSystem:**CipSystem** ▸ LampCluster:**Cluster** ▸ Button:**Process** ▸ normal:**M**

typeOf	**ch.actifsource.solution.cip.generic.core.meta.mode.Transition**
id	1
comment	
state	Pushed
input	Push
outpulse	On
outportMessage	

new ch.actifsource.solution.cip.generic.core.meta.process Outpulse

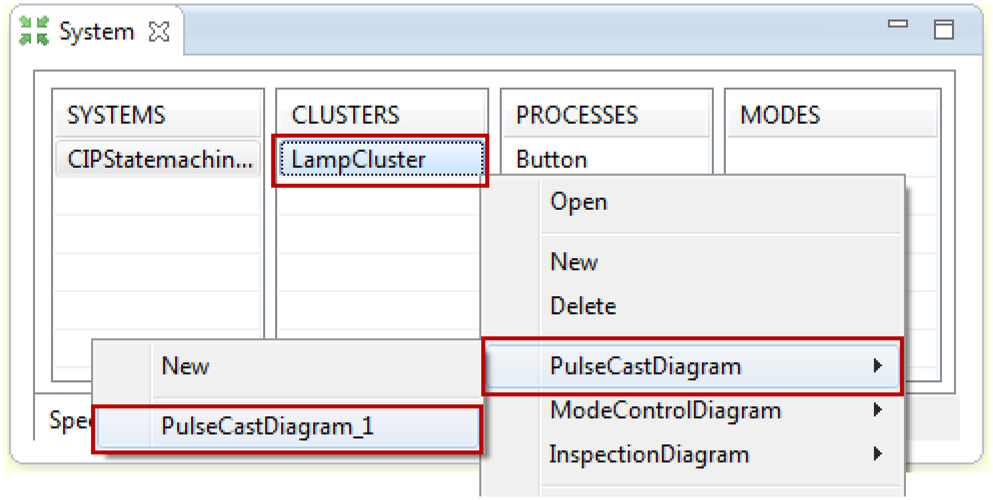
- ⓘ Use **<u>Messages</u>** to communicate with **<u>Processes</u>** from the outer world
- ⓘ Use **<u>Pulses</u>** to communicate between **<u>Processes</u>** within the same **<u>Cluster</u>**
- Add the **<u>Outpulse</u>** On to **<u>Transition</u>** from **<u>State</u>** <u>Released</u> to **<u>State</u>** <u>Pushed</u>
  - Ctrl-Klick on the italics label *Outpulse*
  - Enter the desired **<u>Outpulse</u>** named <u>On</u> and press Ctrl-Space
    - Select an existing **<u>Outpulse</u>**

      or
    - Create a new **<u>Outpulse</u>** (as we do in our example)
- Add the **<u>Outpulse</u>** <u>Off</u> to **<u>Transition</u>** from **<u>State</u>** <u>Pushed</u> to **<u>State</u>** <u>Released</u>



- Set the *Init-State* to specify the **State** to start with
  - Right-Click on **State** Released
  - Shape Actions/Set Init-State

Specify the State Machine



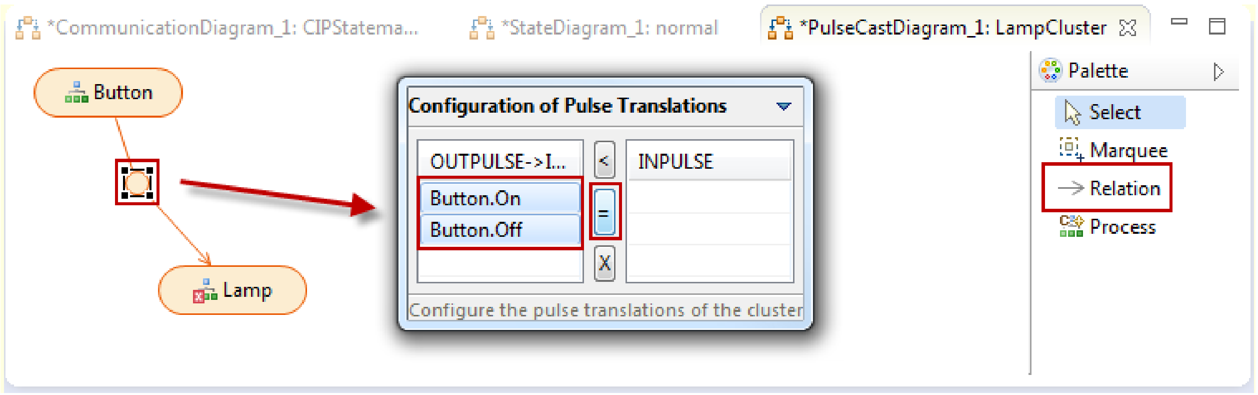
ⓘ Use **Pulses** to communicate between **Processes** within the same **Cluster**

ⓘ The *PulseCastDiagram* specifies which **Processes** sends **Pulses** to which other **Processes**

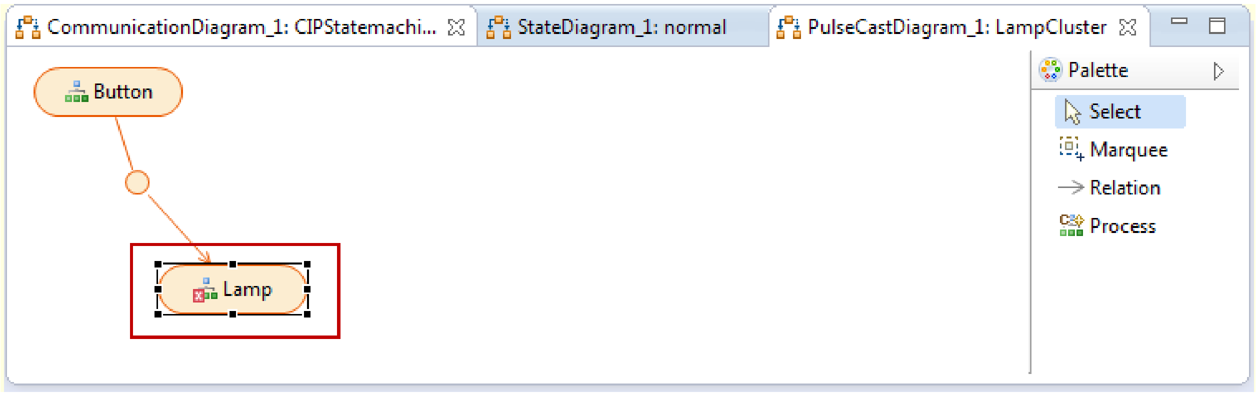
↳ Open the *PulseCastDiagram* for **Cluster** Lamp

- On the *System View* Right-Click on System1.LampCluster
- Select *PulseCastDiagram*
- Select *PulseCastDiagram_1 (or create a new one if necessary)*

Specify the State Machine

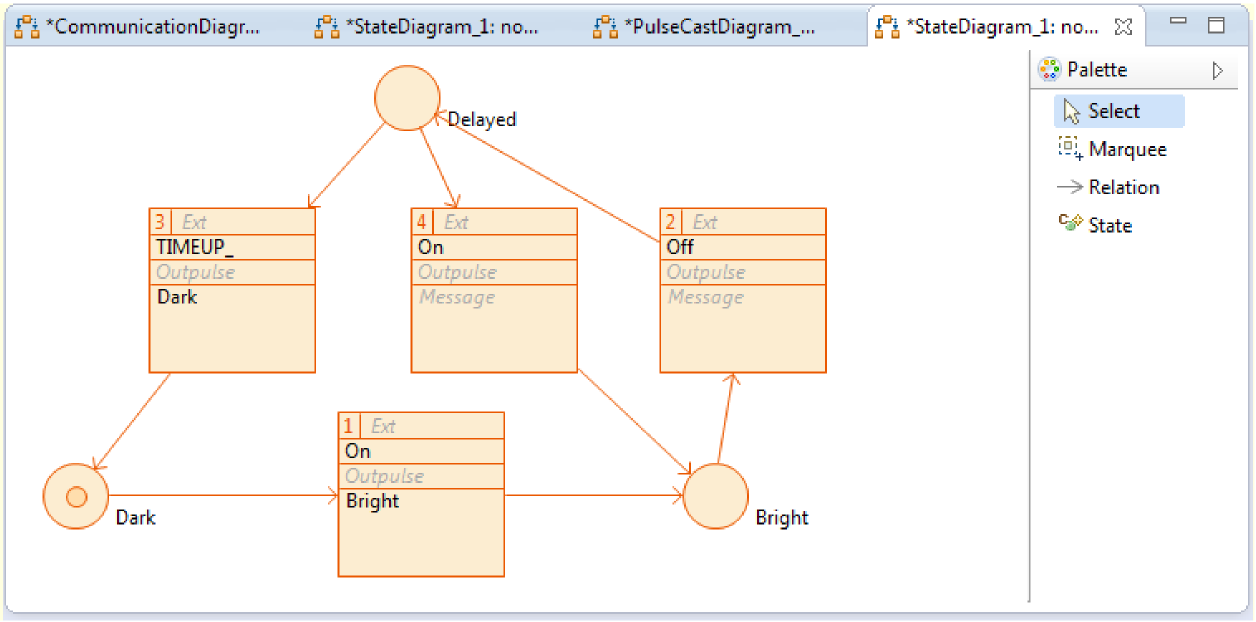

*CommunicationDiagram_1: CIPStatema...
*StateDiagram_1: normal
*PulseCastDiagram_1: LampCluster
Button
Lamp
Configuration of Pulse Translations
OUTPULSE->I...
Button.On
Button.Off
<
=
X
INPULSE
Configure the pulse translations of the cluster
Palette
Select
Marquee
Relation
Process

- Add the **Process** Button and the **Process** Lamp to your *PulseCastDiagram*
  - Right-Click on the Diagram
  - Show Resource
  - Select Button and Lamp
- Connect the **Process** Button and the **Process** Lamp using the *Relation* Tool from the *Palette*
- Configure the *Pulse Translation* between the **Process** Button and the **Process** Lamp
  - Double-Click on the circle of the relation between the the **Process** Button and the **Process** Lamp
  - Select the **Outpulses** Button.On and Button.Off
  - Use the **=**-Tool to create the corresponding **Inpulses** on the **Process** Lamp


CommunicationDiagram_1: CIPStatemachi...
StateDiagram_1: normal
PulseCastDiagram_1: LampCluster
Button
Lamp
Palette
Select
Marquee
Relation
Process

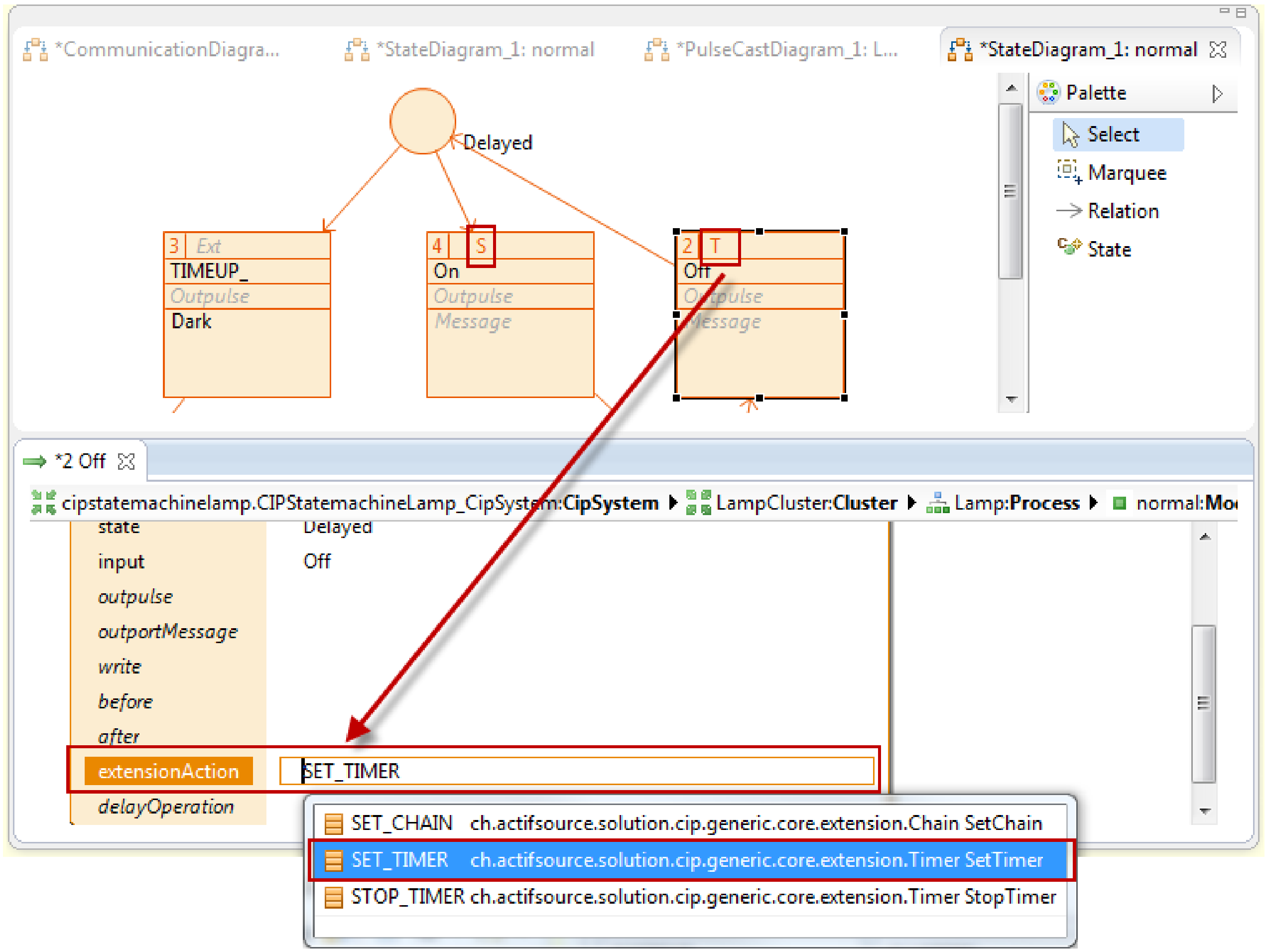
↳ Double-Click on **<u>Process</u>** <u>Lamp</u> to open the *StateDiagram*

Specify the State Machine



- Open the *StateDiagram* for **Process** Lamp (if not already done)
  - On the *System View* Right-Click on System1.Lamp.Button.normal
  - Select *StateDiagram*
  - Select StateDiagram_1
- Create **States** Dark, Bright and Delayed
- Make **State** Dark the *Init-State*
- Create Transition as shown above
  - Select **Pulses** which have been created in the *Pulse Translation*
  - _TIMEUP is a special pulses emitted if timer expires
  - Select Lamp **Messages** to the outer world

Specify the State Machine

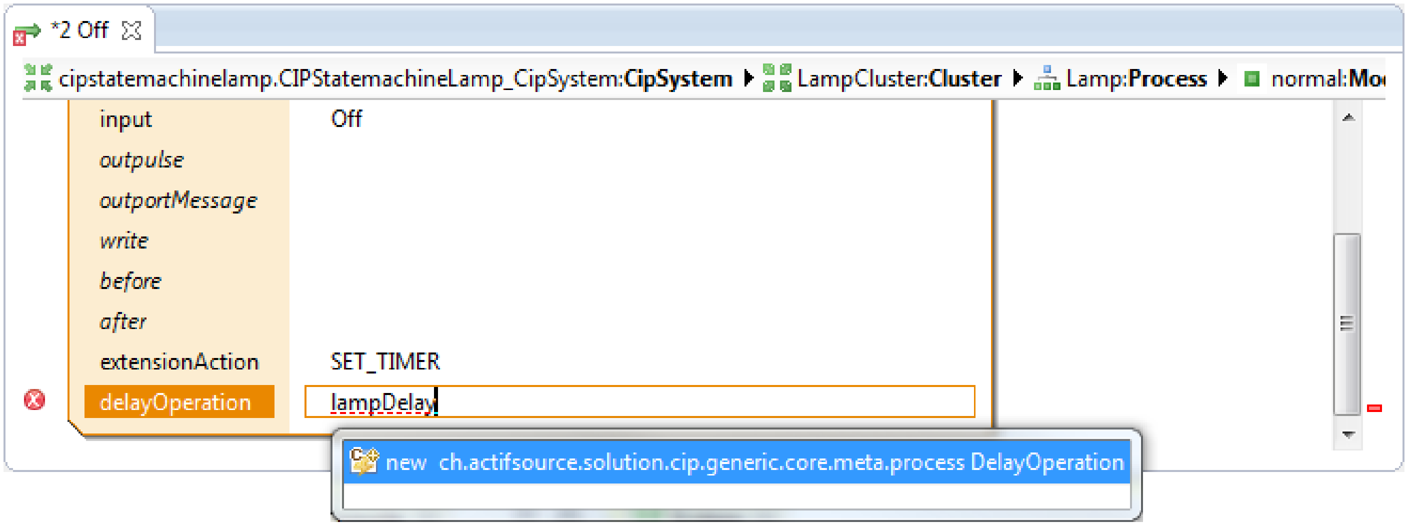


state	Delayed
input	Off
outpulse	
outportMessage	
write	
before	
after	
extensionAction	SET_TIMER
delayOperation	

SET_CHAIN	ch.actifsource.solution.cip.generic.core.extension.Chain SetChain
SET_TIMER	ch.actifsource.solution.cip.generic.core.extension.Timer SetTimer
STOP_TIMER	ch.actifsource.solution.cip.generic.core.extension.Timer StopTimer

- Start Timer in Transition #2
  - Ctrl-Click in label *Ext*
  - Select SET_TIMER
  - Specify DelayOperation (see next page)
- Stop Timer in Transition #4
  - Ctrl-Click in label *Ext*
  - Select STOP_TIMER

Specify the State Machine



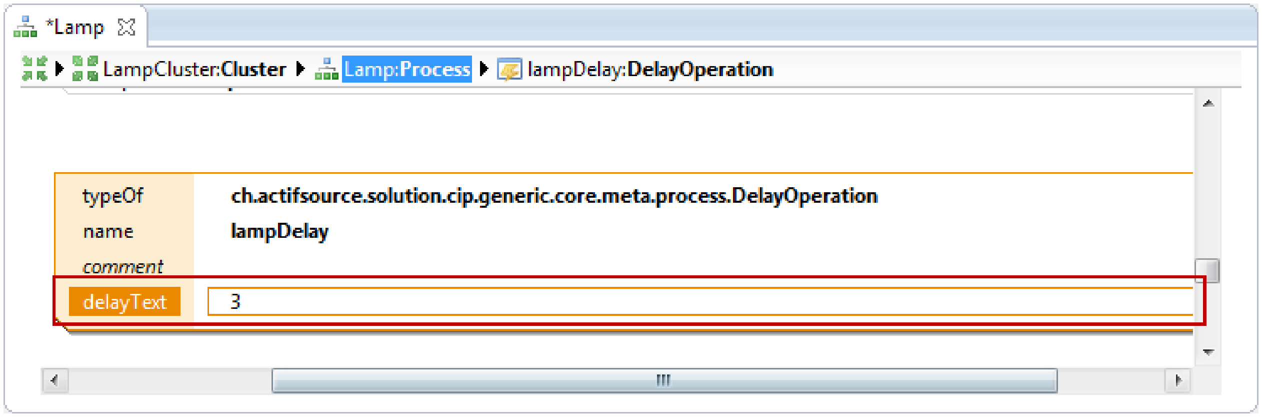
*2 Off

cipstatemachinelamp.CIPStatemachineLamp_CipSystem:**CipSystem** ▸ LampCluster:**Cluster** ▸ Lamp:**Process** ▸ normal:**Mo**

input	Off
outpulse	
outportMessage	
write	
before	
after	
extensionAction	SET_TIMER
delayOperation	lampDelay

new ch.actifsource.solution.cip.generic.core.meta.process DelayOperation

- Create a DelayOperation called lampDelay
- The DelayOperation shall return the delay in system ticks



*Lamp

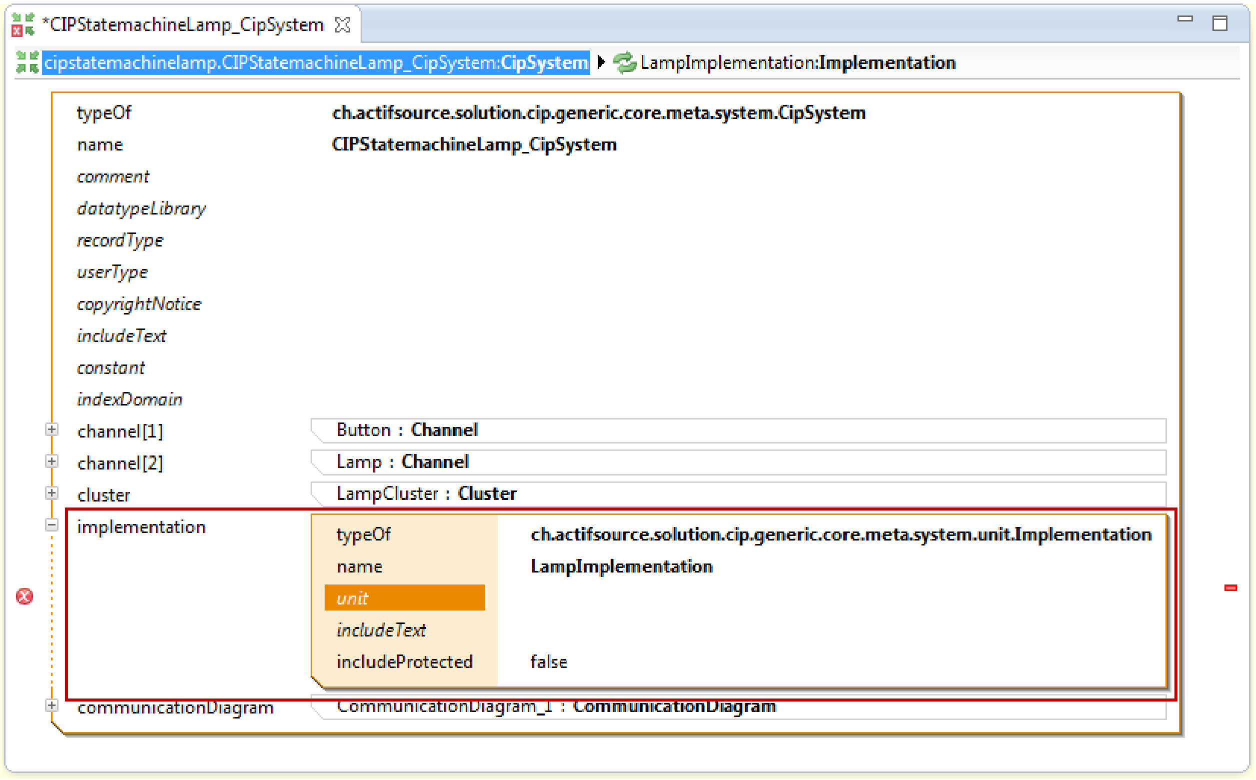
LampCluster:**Cluster** ▸ Lamp:**Process** ▸ lampDelay:**DelayOperation**

typeOf	**ch.actifsource.solution.cip.generic.core.meta.process.DelayOperation**
name	**lampDelay**
comment	
delayText	3

Part IV: Generating State Machine Code

- We have to define the details for the state machine implementation to generate the code

Generating State Machine Code



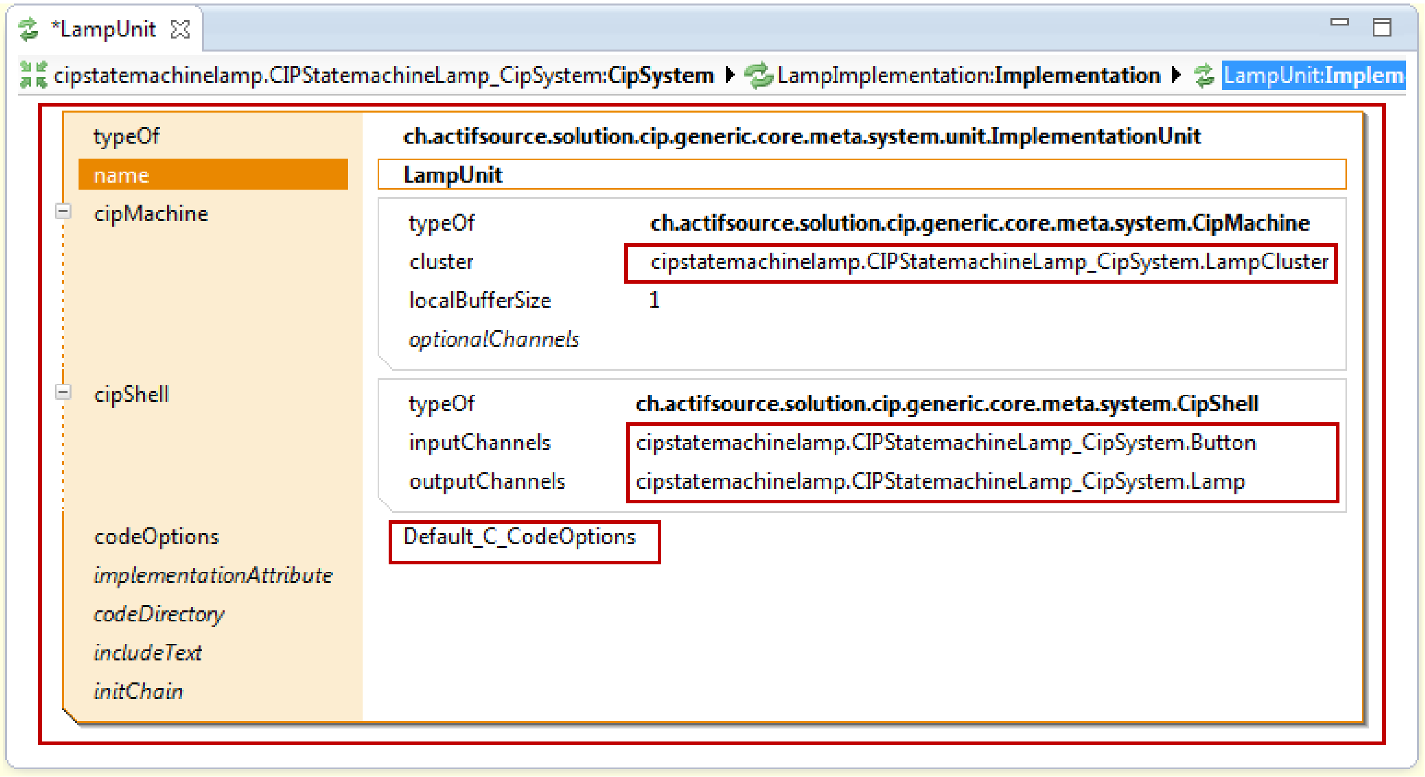
*CIPStatemachineLamp_CipSystem

cipstatemachinelamp.CIPStatemachineLamp_CipSystem:**CipSystem** ▸ LampImplementation:**Implementation**

typeOf	**ch.actifsource.solution.cip.generic.core.meta.system.CipSystem**
name	**CIPStatemachineLamp_CipSystem**
comment	
datatypeLibrary	
recordType	
userType	
copyrightNotice	
includeText	
constant	
indexDomain	
channel[1]	Button : **Channel**
channel[2]	Lamp : **Channel**
cluster	LampCluster : **Cluster**
implementation	typeOf: **ch.actifsource.solution.cip.generic.core.meta.system.unit.Implementation**; name: **LampImplementation**; *unit*; *includeText*; includeProtected: false
communicationDiagram	CommunicationDiagram_1 : **CommunicationDiagram**

- Create a new *Implementation* for **System** System1
  - On the *System View* Right-Click on System1
  - Create a new Implementation

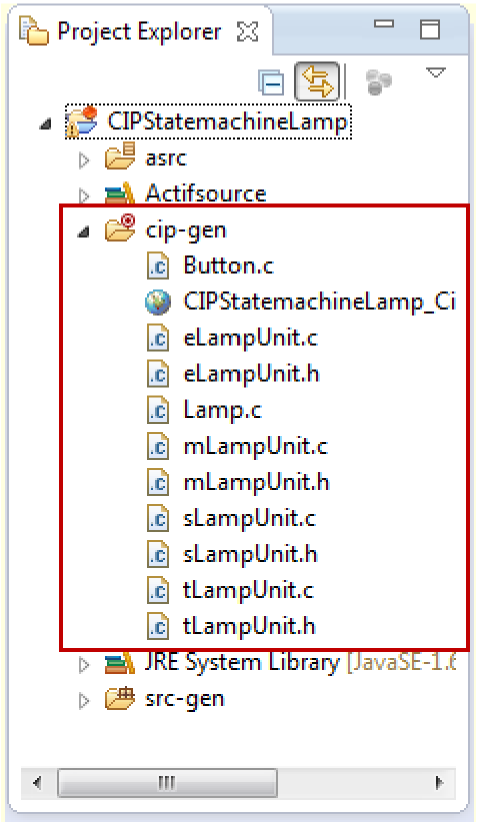
Generating State Machine Code


*LampUnit
cipstatemachinelamp.CIPStatemachineLamp_CipSystem:CipSystem
LampImplementation:Implementation
LampUnit:Implem

typeOf	ch.actifsource.solution.cip.generic.core.meta.system.unit.ImplementationUnit
name	LampUnit
cipMachine	typeOf: ch.actifsource.solution.cip.generic.core.meta.system.CipMachine cluster: cipstatemachinelamp.CIPStatemachineLamp_CipSystem.LampCluster localBufferSize: 1 *optionalChannels*
cipShell	typeOf: ch.actifsource.solution.cip.generic.core.meta.system.CipShell inputChannels: cipstatemachinelamp.CIPStatemachineLamp_CipSystem.Button outputChannels: cipstatemachinelamp.CIPStatemachineLamp_CipSystem.Lamp
codeOptions	Default_C_CodeOptions
implementationAttribute	
codeDirectory	
includeText	
initChain	

↳ Create a new *Implementation* as shown above

Generating State Machine Code



```
CIPStatemachineLamp
  asrc
  Actifsource
  cip-gen
    Button.c
    CIPStatemachineLamp_Ci
    eLampUnit.c
    eLampUnit.h
    Lamp.c
    mLampUnit.c
    mLampUnit.h
    sLampUnit.c
    sLampUnit.h
    tLampUnit.c
    tLampUnit.h
  JRE System Library [JavaSE-1.6
  src-gen
```

ⓘ Find the generated code in the folder *cip-gen*

ⓘ Find the documentation (html) code in the folder *cip-gen*