Table of Contents

Organizer View	1
Package DataTypes	3
Package Utils	5
Package Actuators	7
Block Type TaskLight	8
Process TaskLight	9
Block Type Dimmer	10
Process Dimmer	11
Block Type PulsGen	12
Process PulsGen	13

SDT	rw	/SFB_501/teams/team3/Basis−Projekt.BaX/Entwicklung/RequirementsSp
	rw	/SFB_501/teams/team3/Basis−Projekt.BaX/Entwicklung/RequirementsSp
Used Packages		
DataTypes	r−	../Datatypes/LatestRev/DataTypes.sun
Utils	r−	../Utils/LatestRev/Utils.sun
ActuatorPackage		
Actuators	rw	Actuators.sun
TaskLight	rw	TaskLight.sbt
TaskLight	rw	TaskLight.spr
Dimmer	rw	Dimmer.sbt
Dimmer	rw	Dimmer.spr
PulsGen	rw	PulsGen.sbt
PulsGen	rw	PulsGen.spr

ecification/Actuators/Actuator.sdt

ecification/Actuators/

Package Datatypes


```
SIGNAL dummy;
SIGNAL ident (PID, CharString);
```

```
SYNTYPE Analog=Real  ENDSYNTYPE;
SYNTYPE Binary=Integer CONSTANTS 0,1 ENDSYNTYPE;
SYNTYPE Percent=Real CONSTANTS 0:100 ENDSYNTYPE;
SYNTYPE Day=Integer CONSTANTS 1:365 ENDSYNTYPE;
SYNTYPE WkDay=Integer CONSTANTS 1:7 ENDSYNTYPE;
SYNTYPE Celsius=Real CONSTANTS -60:100 ENDSYNTYPE;
SYNTYPE Lux=REAL CONSTANTS -1:120000 ENDSYNTYPE;
SYNTYPE HallwayIdType = Integer CONSTANTS 1:3 ENDSYNTYPE;
SYNTYPE DegF=Real CONSTANTS -60:200 ENDSYNTYPE; /*Fahrenheit*/
SYNTYPE GPM=Real CONSTANTS 0:200 ENDSYNTYPE; /*Gallon per minute*/
SYNTYPE Psi=Real CONSTANTS 0:200 ENDSYNTYPE; /*Pound per square inch*/
SYNTYPE Watt=Real CONSTANTS 0:10000 ENDSYNTYPE; /*Power unit*/
SYNTYPE ErrorType = INTEGER CONSTANTS 0:100 ENDSYNTYPE;
SYNTYPE Joule = REAL CONSTANTS 0:3000000 ENDSYNTYPE;
SYNTYPE DayTime = INTEGER ENDSYNTYPE;
SYNTYPE LightSourceType = INTEGER CONSTANTS 1:3 ENDSYNTYPE;
```

```
NEWTYPE RoomIdType
struct
rId CHARSTRING;
sId CHARSTRING;
ENDNEWTYPE;
```

```
NEWTYPE LightSourceListType
ARRAY (INTEGER, LightSourceType)
ENDNEWTYPE;
```

```
/* Fehlt: Die Prioritaetsliste fuer die Lichtquellen */
NEWTYPE LightScene
struct
name CharString;
rIll Lux;
dIll Lux;
ls LightSourceListType;
ENDNEWTYPE;
```

```
NEWTYPE CalTimeType
struct
year, month, day, hour, min, sec INTEGER;
operators
   toTime: CalTimeType -> Time;
   fromTime: Time->CalTimeType;
/*#ADT (B)*/
ENDNEWTYPE;
```

Package Datatypes

```
SIGNAL getIlluminance ;
SIGNAL newIlluminance (RoomIdType, Lux);
```

```
SIGNAL newHwEnergyConsumption (Joule, Duration);
SIGNAL newHwMalFunction( Errortype );
SIGNAL getHwPBDelay;
SIGNAL getHwOccDelay;

SIGNAL setHwEnergyDiffTime(Duration);
SIGNAL setHwLight( Boolean );
SIGNAL setHwPbDelay(Duration);
SIGNAL setHwOccDelay (Duration);

SIGNAL setRoomDelay (RoomIdType, Duration);
SIGNAL setRoomReoccDelay (RoomIdType, Duration);
SIGNAL setDefaultLightScene (RoomIdType, LightScene);
SIGNAL setAllDefaultLightScene (LightScene);
SIGNAL setDefaultAmbientLightLevel (RoomIdType, Lux);
SIGNAL setRoomLight (RoomIdType, Boolean);
SIGNAL setRoomEnergyDiffTime (Duration);
SIGNAL newRoomMalfunction (RoomIdType, ErrorType);
SIGNAL newRoomEnergyConsumption (RoomIdType, Joule, Duration);

SIGNAL newEntrance;
```

```
USE DataTypes;
```

Package Utils


```
/*setxxValue: root sets value in remote node*/
/*pollxxValue: root polls value of remote node*/
/*putxxValue: remote node puts value into root*/

REMOTE PROCEDURE setPhysVal; FPAR IN  Analog;
REMOTE PROCEDURE putAnalogVal; FPAR IN  PID, Analog;
REMOTE PROCEDURE putBinaryVal; FPAR IN  PID, Binary;
REMOTE PROCEDURE pollPhysVal; FPAR IN  PID;
```

```
procedure toJoule;
fpar IN w Watt,
IN dimV Percent,
IN timeDiff Duration;
RETURNS Joule;
EXTERNAL;
```

```
/* Proz zum Ausdruck eines beliebigen
Binaerwerts mit der aktuellen Zeit */
procedure printBi;
fpar IN c Charstring,
IN b Binary,
IN dur Time;
EXTERNAL;
```

```
/* Proz zum Ausdruck eines beliebigen
Anaolgwerts mit der aktuellen Zeit */
procedure printAn;
fpar IN c CharString,
IN a Analog,
IN dur Time;
EXTERNAL;
```

```
/* Proz zum Umformen der Kalendereit in die aktuelle Zeit*/
/*procedure calTimeToTime;
fpar IN actTime calTimeType;
RETURNS Time;
EXTERNAL;*/
```

```
/* Proz zum Umformen der aktuellen Zeit in Kalenderzeit*/
/*procedure timeToCalTime;
fpar IN actTime Time;
RETURNS CalTimeType;
EXTERNAL;*/
```

```
/* Proz zum Ausdruck eines beliebigen
Strings mit der aktuellen Zeit */
procedure printIt;
fpar IN c CharString,
IN dur Time;
EXTERNAL;
```

USE DataTypes;

Package Utils

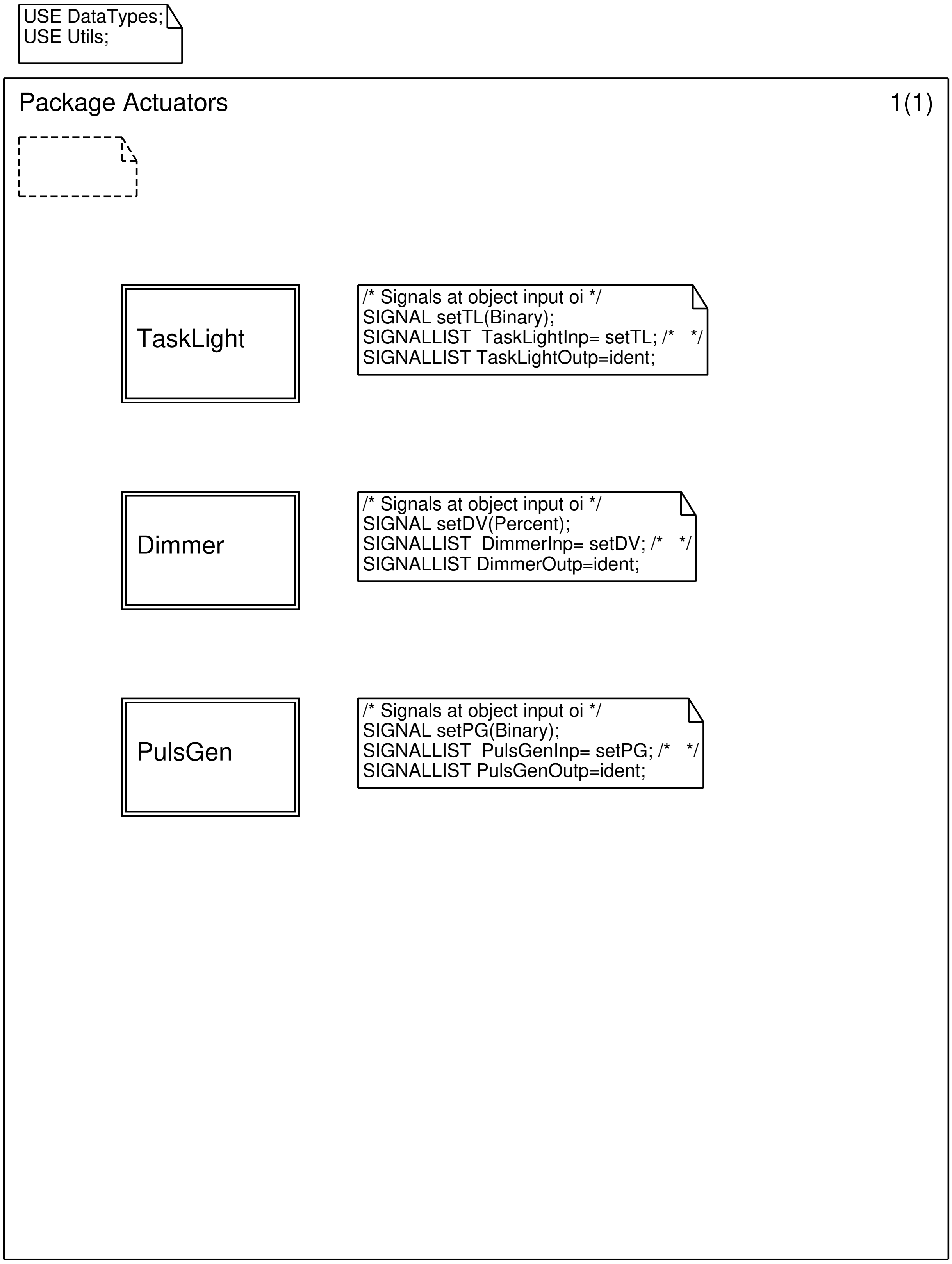
/* Proz zum Verbindungsaufbaus des Systems
mit der Umwelt (physSock)*/
procedure initSockets;
EXTERNAL;

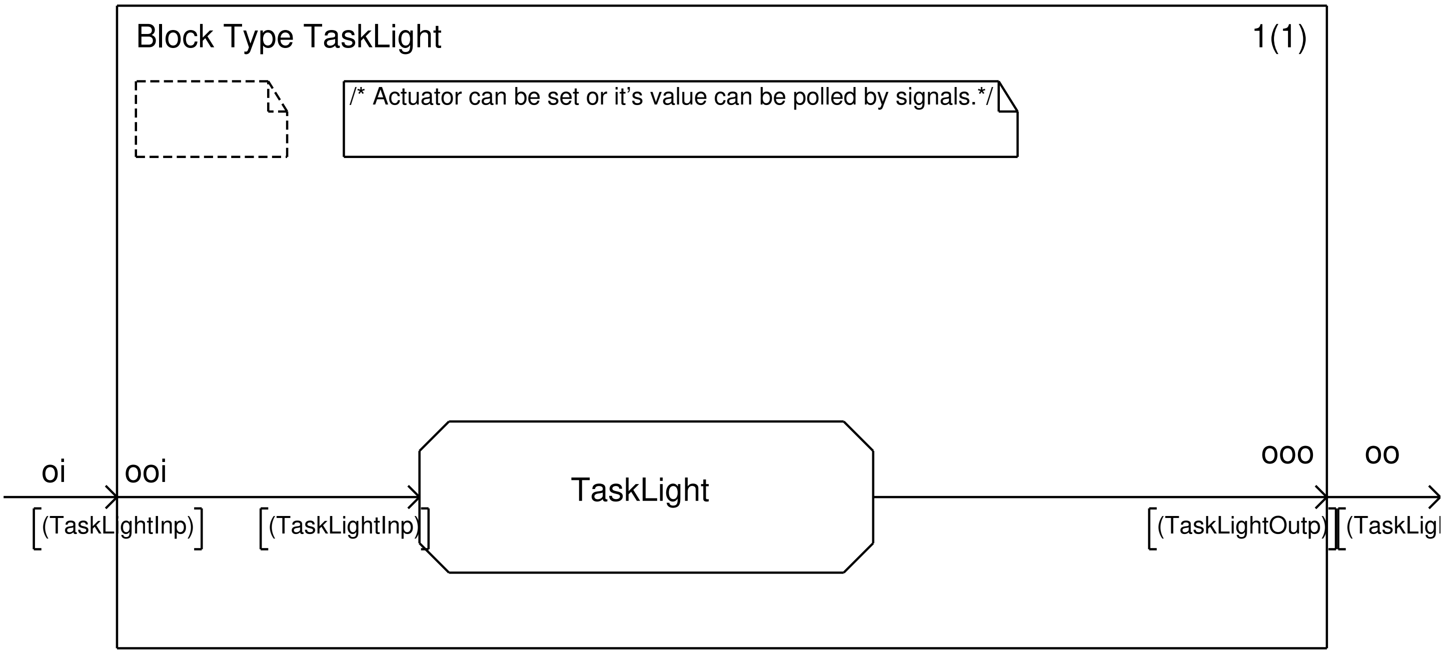
/* Proz., die periodisch einen eventuellen Eingang an dem beiden
Socket physSock bearbeitet.*/
procedure checkSockets;
returns INTEGER;
EXTERNAL;

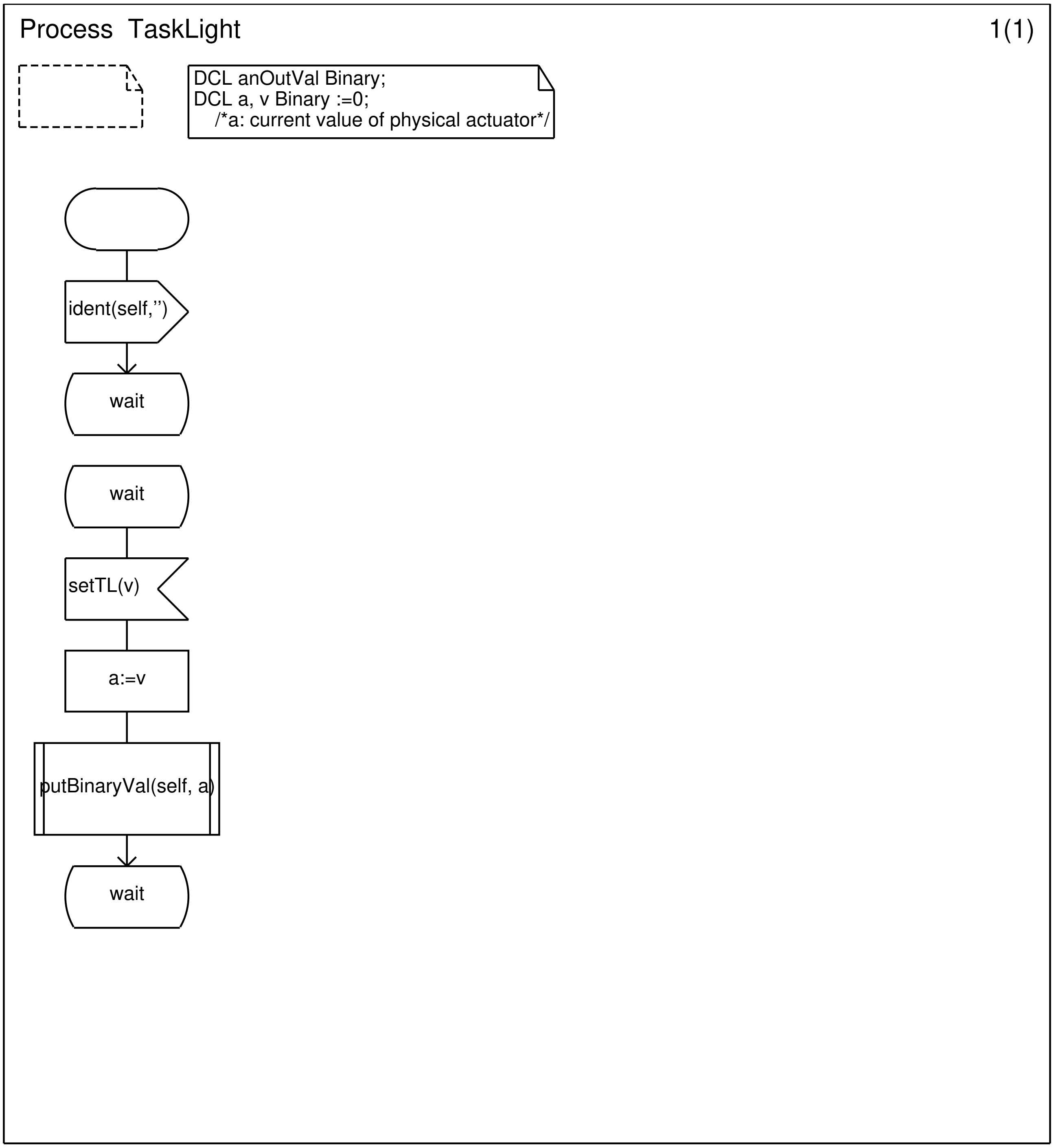
/* Proz., die einen binaeren physik. Werte setzen.*/
procedure setBinOutVal;
FPAR IN actId CharString,
IN val Binary;
EXTERNAL;

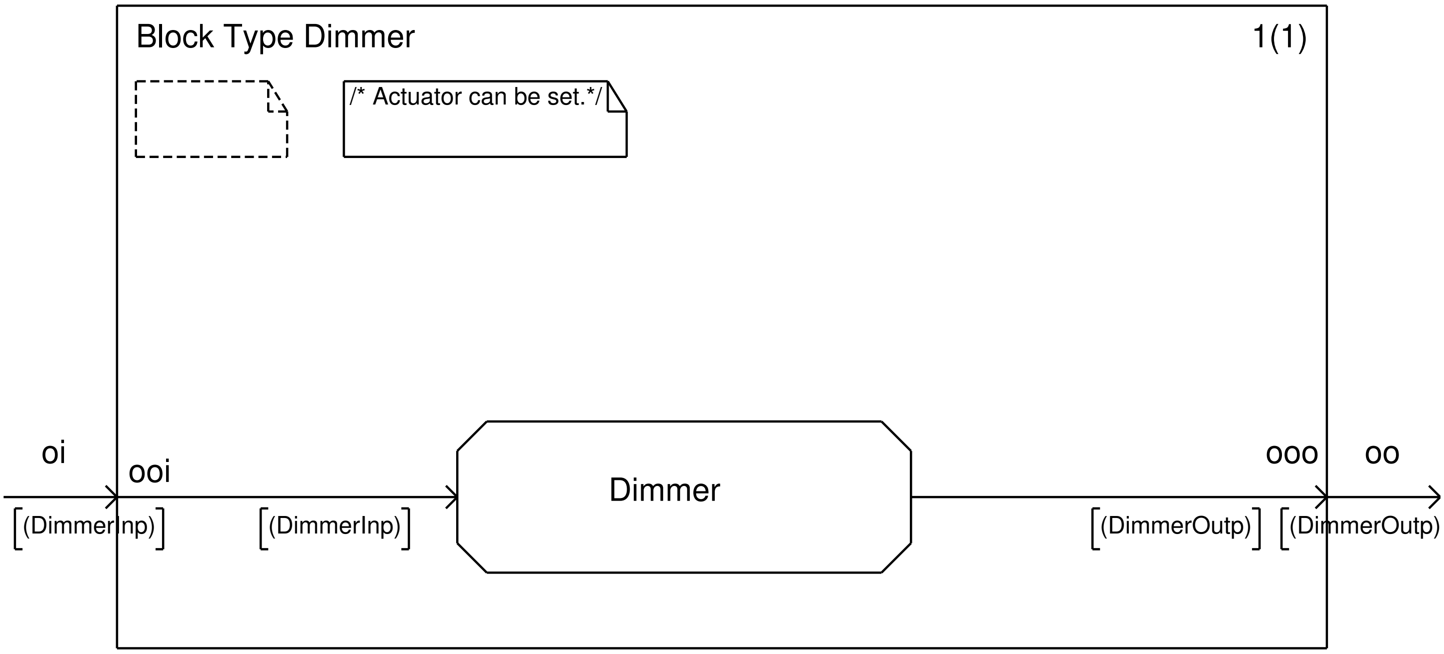
/* Proz., die einen analogen physik. Werte setzen.*/
procedure setAnOutVal;
FPAR IN actId CharString,
IN val Analog;
EXTERNAL;

/* Proz., die einen physik. Werte anfordert.*/
procedure reqPhysVal;
FPAR IN actId CharString;
EXTERNAL;

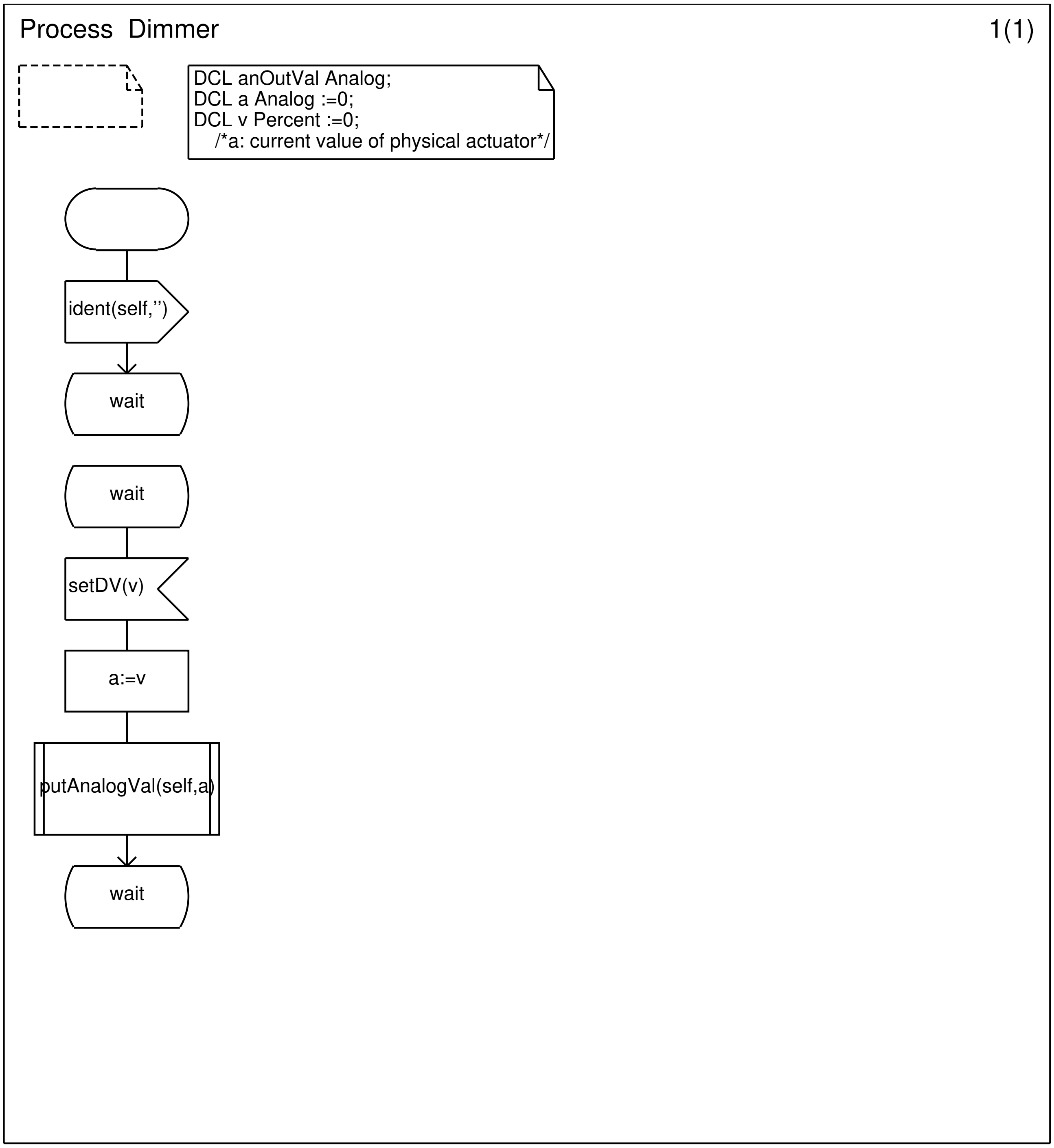

USE DataTypes;
USE Utils;
Package Actuators
1(1)
TaskLight
/* Signals at object input oi */
SIGNAL setTL(Binary);
SIGNALLIST TaskLightInp= setTL; /* */
SIGNALLIST TaskLightOutp=ident;
Dimmer
/* Signals at object input oi */
SIGNAL setDV(Percent);
SIGNALLIST DimmerInp= setDV; /* */
SIGNALLIST DimmerOutp=ident;
PulsGen
/* Signals at object input oi */
SIGNAL setPG(Binary);
SIGNALLIST PulsGenInp= setPG; /* */
SIGNALLIST PulsGenOutp=ident;

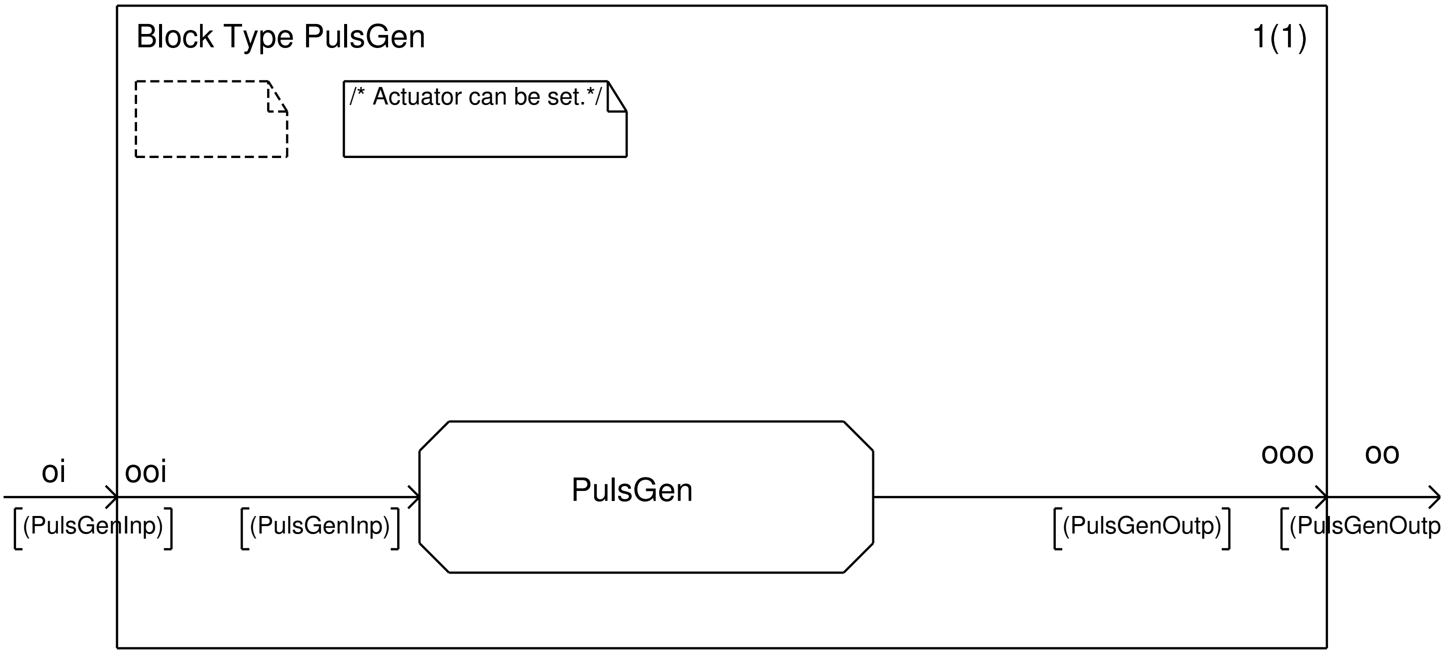

Block Type TaskLight
1(1)
/* Actuator can be set or it's value can be polled by signals.*/
TaskLight
oi
ooi
(TaskLightInp)
(TaskLightInp)
ooo
oo
(TaskLightOutp)
(TaskLig

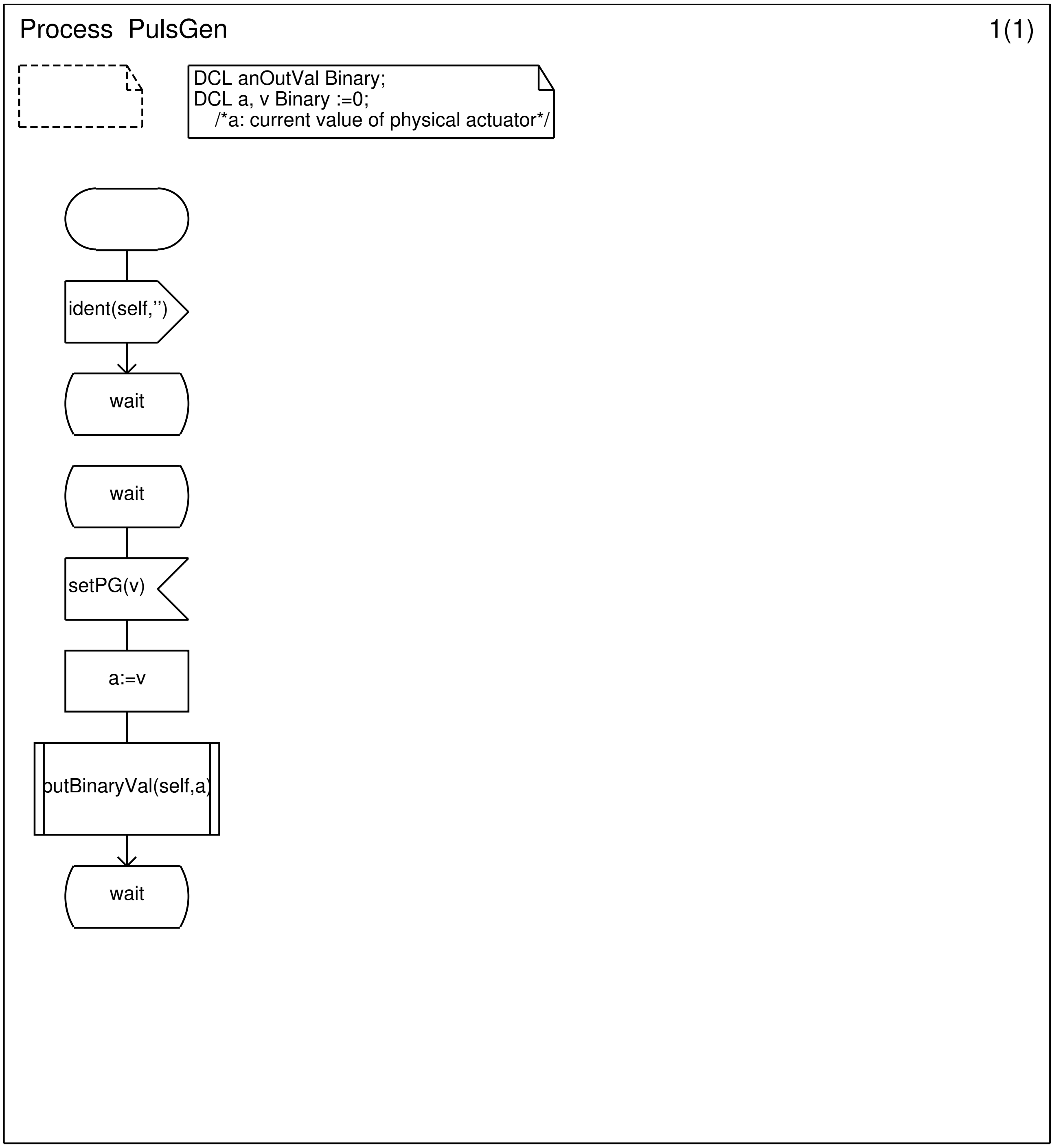

Process TaskLight
1(1)
DCL anOutVal Binary;
DCL a, v Binary :=0;
/*a: current value of physical actuator*/
ident(self,'')
wait
wait
setTL(v)
a:=v
putBinaryVal(self, a)
wait



Block Type Dimmer
1(1)
/* Actuator can be set.*/
oi
ooi
(DimmerInp)
(DimmerInp)
Dimmer
ooo
oo
(DimmerOutp)
(DimmerOutp)


Process Dimmer
1(1)
DCL anOutVal Analog;
DCL a Analog :=0;
DCL v Percent :=0;
/*a: current value of physical actuator*/
ident(self,'')
wait
wait
setDV(v)
a:=v
putAnalogVal(self,a)
wait


Block Type PulsGen
1(1)
/* Actuator can be set.*/
PulsGen
oi
ooi
(PulsGenInp)
(PulsGenInp)
ooo
oo
(PulsGenOutp)
(PulsGenOutp


Process PulsGen
1(1)
DCL anOutVal Binary;
DCL a, v Binary :=0;
/*a: current value of physical actuator*/
ident(self,'')
wait
wait
setPG(v)
a:=v
outBinaryVal(self,a)
wait