

## Logikai hálózat szimulálórendszer

Készítette Doxygen 1.13.2



<b>1. Felhasználói dokumentáció</b>	<b>1</b>
1.0.1. Program leírása	1
1.0.2. Szimuláció működése	1
1.0.3. Hálózat létrehozása	2
1.1. Programozói dokumentáció	2
1.1.1. Hibák, kivételek	2
1.1.2. Tesztesetek	2
1.1.3. Egyéb megjegyzések	2
1.1.4. Új komponens felvétele	2
1.2. UML diagram	3
<b>2. Hierarchikus mutató</b>	<b>5</b>
2.1. Osztályhierarchia	5
<b>3. Osztálymutató</b>	<b>7</b>
3.1. Osztálylista	7
<b>4. Fájlmutató</b>	<b>9</b>
4.1. Fájllista	9
<b>5. Osztályok dokumentációja</b>	<b>11</b>
5.1. AND osztályreferencia	11
5.1.1. Konstruktorkok és destruktorok dokumentációja	13
5.1.1.1. AND()	13
5.1.2. Tagfüggvények dokumentációja	13
5.1.2.1. get_name()	13
5.1.2.2. write()	13
5.2. Component osztályreferencia	14
5.2.1. Tagfüggvények dokumentációja	15
5.2.1.1. write()	15
5.3. INP osztályreferencia	15
5.3.1. Konstruktorkok és destruktorok dokumentációja	17
5.3.1.1. INP()	17
5.3.2. Tagfüggvények dokumentációja	17
5.3.2.1. get_name()	17
5.3.2.2. write()	17
5.4. LogicNetwork osztályreferencia	18
5.4.1. Konstruktorkok és destruktorok dokumentációja	20
5.4.1.1. LogicNetwork()	20
5.4.2. Tagfüggvények dokumentációja	20
5.4.2.1. add_component()	20
5.4.2.2. bulk_update()	20
5.4.2.3. get_os()	21
5.4.2.4. get_wire()	21

5.5. LogicNetworkConfigurer osztályreferencia . . . . .	22
5.5.1. Tagfüggvények dokumentációja . . . . .	24
5.5.1.1. read_logic_network() . . . . .	24
5.5.1.2. write_logic_network() . . . . .	24
5.6. NOT osztályreferencia . . . . .	25
5.6.1. Konstruktorok és destruktorok dokumentációja . . . . .	27
5.6.1.1. NOT() . . . . .	27
5.6.2. Tagfüggvények dokumentációja . . . . .	27
5.6.2.1. get_name() . . . . .	27
5.6.2.2. write() . . . . .	27
5.7. OR osztályreferencia . . . . .	28
5.7.1. Konstruktorok és destruktorok dokumentációja . . . . .	30
5.7.1.1. OR() . . . . .	30
5.7.2. Tagfüggvények dokumentációja . . . . .	30
5.7.2.1. get_name() . . . . .	30
5.7.2.2. write() . . . . .	30
5.8. PRINT osztályreferencia . . . . .	31
5.8.1. Konstruktorok és destruktorok dokumentációja . . . . .	33
5.8.1.1. PRINT() . . . . .	33
5.8.2. Tagfüggvények dokumentációja . . . . .	33
5.8.2.1. get_name() . . . . .	33
5.8.2.2. write() . . . . .	33
5.9. STD_INP osztályreferencia . . . . .	34
5.9.1. Konstruktorok és destruktorok dokumentációja . . . . .	36
5.9.1.1. STD_INP() . . . . .	36
5.9.2. Tagfüggvények dokumentációja . . . . .	36
5.9.2.1. get_name() . . . . .	36
5.9.2.2. update() . . . . .	36
5.9.2.3. write() . . . . .	36
5.10. Wire osztályreferencia . . . . .	37
5.10.1. Tagfüggvények dokumentációja . . . . .	37
5.10.1.1. get_signal() . . . . .	37
5.10.1.2. set_signal() . . . . .	38
5.11. XOR osztályreferencia . . . . .	38
5.11.1. Konstruktorok és destruktorok dokumentációja . . . . .	40
5.11.1.1. XOR() . . . . .	40
5.11.2. Tagfüggvények dokumentációja . . . . .	40
5.11.2.1. get_name() . . . . .	40
5.11.2.2. write() . . . . .	40
<b>6. Fájlok dokumentációja . . . . .</b>	<b>41</b>
6.1. basic_components.h . . . . .	41

6.2. AND.h	41
6.3. INP.h fájlreferencia	41
6.3.1. Részletes leírás	42
6.4. INP.h	43
6.5. NOT.h fájlreferencia	43
6.5.1. Részletes leírás	44
6.6. NOT.h	44
6.7. OR.h fájlreferencia	45
6.7.1. Részletes leírás	46
6.8. OR.h	46
6.9. PRINT.h fájlreferencia	46
6.9.1. Részletes leírás	47
6.10. PRINT.h	48
6.11. STD_INP.h fájlreferencia	48
6.11.1. Részletes leírás	49
6.12. STD_INP.h	49
6.13. XOR.h fájlreferencia	50
6.13.1. Részletes leírás	51
6.14. XOR.h	51
6.15. component.h fájlreferencia	51
6.15.1. Részletes leírás	52
6.16. component.h	52
6.17. logic_network.h fájlreferencia	52
6.17.1. Részletes leírás	53
6.18. logic_network.h	54
6.19. logic_network_configurer.h fájlreferencia	54
6.19.1. Részletes leírás	55
6.20. logic_network_configurer.h	56
6.21. main.cpp fájlreferencia	56
6.21.1. Részletes leírás	56
6.22. wire.h fájlreferencia	56
6.22.1. Részletes leírás	57
6.23. wire.h	57
<b>Tárgymutató</b>	<b>59</b>



# 1. fejezet

## Felhasználói dokumentáció

### 1.0.1. Program leírása

A program egy logikai hálózatot modellez. Ebben komponensek vannak, amiknek több bemenete, de csak egy kimenete van (például **AND** kapu). A be és kimenetek vezetékekre csatlakoznak, amik a jelet ütemezve továbbítják a többi komponens felé.

A programban megvalósuló osztályok:

- **AND, XOR, OR, NOT** bináris kapuk
- **Wire**: Komponenseket összekötő vezeték
- **INP**: Konstans jelet adó komponens
- **STD\_INP**: Olyan **INP**, ami a jelet első frissítéskor a standard bemenetről olvassa be
- **LogicNetwork**: A vezetékeket és komponenseket egybefoglaló logikai hálózat osztály
- **LogicNetworkConfigurer**: Olyan logikai hálózat osztály, aminek a felépítését fájlból lehet beolvasni és visszaírni

### 1.0.2. Szimuláció működése

Az időbeli szimulálás miatt a vezetékeknek két része van. A vezeték egyes részeire akárhány komponens csatlakozhat, így csomópontként is funkcionál. A vezeték „elejére” csak írni lehet, a „végéről” pedig csak olvasni lehet a jelet.

A hálózat frissítése („órakeze”) két fázisból áll. Első lépésben a komponensek leolvassák a bemenetükre kötött vezetékek végéről az adatot. Az eredmény kirakják a rájuk kötött vezetékek elejére. A nagyobb jelszint felülírja a kisebbet.

A második fázisban minden vezeték átrakja a jelet az elejéről a végére. Így a frissítés sorrendjétől független lesz a hálózat viselkedése.

Ennek megfelelően egy egész logikai hálózat frissítése először az összes komponens, majd az összes vezeték frissítéséből áll. (Ezt valósítja meg a logikai hálózat osztály `update()` függvénye)

A logikai hálózatnak van `bulk_update()` függvénye, ami egymás után több frissítést végez el. Az egyes frissítések kezdetét a kimeneten jelzi.

### 1.0.3. Hálózat létrehozása

Hálózatot a [main.cpp](#) fájl példái alapján lehet létrehozni kódban a logikai hálózatok tagfüggvényeivel, vagy pedig szöveges fájl alapján is. A fájl szerinti beolvasásra is van példa a [main.cpp](#)-ben.

Hálózat létrehozásakor meg kell adni a kábelek számát, és az adatfolyamot ahova a frissítéseket kezdetét jelző szöveget írni fogja. Fájlból olvasásnál a [PRINT](#) komponensek is ide írnak.

A fájlból beolvasásnál az első sorban a kábelek, majd a komponensek számát kell megadni, utána soronként az egyes komponenseket a nekik megfelelő formátumban. A bemenetek a kábel sorszámát jelzik a hálózatban (0-tól indexelve):

- [AND](#) / [OR](#) / [XOR](#) [1. bemenet] [2. bemenet] [kimenet]
- [NOT](#) [bemenet] [kimenet]
- [INP](#) [kimenet] [kimeneti jel]
- [STD\\_INP](#) [kimenet] [tetszőleges címke]
- [PRINT](#) [bemenet] [tetszőleges címke]

## 1.1. Programozói dokumentáció

### 1.1.1. Hibák, kivételek

- A logikai hálózat beolvasása ha nem sikerült, szövegesen eldobja, hol érzékelt hibát.
- A logikai hálózatban a vezetékek lekérésekor szöveges hibát dob, ha nincs elég kábel.
- [STD\\_INP](#) komponens szöveges hibát dob, ha a bejövő jel nem 0 vagy 1. Bár a program tudna más jeleket is kezelni, a megvalósított komponensek csak ezt a két értéket értelmezik.

### 1.1.2. Tesztesetek

A [main.cpp](#) fájlban vannak a `gtest_lite`-al megvalósított tesztek. Az utolsó teszt eset standard bemenetről olvas, amihez példabemenetet kap a `Jporta`.

### 1.1.3. Egyéb megjegyzések

A komponensek mutatókkal tárolják, melyik vezetékekhez kapcsolódnak. Emiatt minden osztálynál le van tiltva a másoló konstruktor és az értékadó operátor, mivel nem lehet / nincs értelme ugyan olyan objektumokat létrehozni.

### 1.1.4. Új komponens felvétele

Az új komponens felvételéhez az osztálynak a többi komponenshez hasonlóan a [Component](#) leszármazottjának kell lennie. Ezen felül tetszőlegesen megvalósítható a beolvasás / kiírás. A beolvasást a [LogicNetworkConfigurer](#) `read_logic_network()` függvényében kell megvalósítani a többihez hasonlóan. A komponens nevében nem szerepelhet whitespace, és legfeljebb 100 karakter hosszú lehet.



## 1.2. UML diagram



## 2. fejezet

# Hierarchikus mutató

### 2.1. Osztályhierarchia

Majdnem (de nem teljesen) betűrendbe szedett leszármazási lista:

Component . . . . .	14
AND . . . . .	11
INP . . . . .	15
NOT . . . . .	25
OR . . . . .	28
PRINT . . . . .	31
STD_INP . . . . .	34
XOR . . . . .	38
LogicNetwork . . . . .	18
LogicNetworkConfigurer . . . . .	22
Wire . . . . .	37



## 3. fejezet

# Osztálymutató

### 3.1. Osztálylista

Az összes osztály, struktúra, unió és interfész listája rövid leírásokkal:

AND	11
Component	14
INP	15
LogicNetwork	18
LogicNetworkConfigurer	22
NOT	25
OR	28
PRINT	31
STD_INP	34
Wire	37
XOR	38



## 4. fejezet

# Fájlmutató

### 4.1. Fájllista

Az összes dokumentált fájl listája rövid leírásokkal:

<a href="#">basic_components.h</a>	41
<a href="#">AND.h</a>	41
<a href="#">INP.h</a>	
Kábelre konstans értéket író komponens	41
<a href="#">NOT.h</a>	
NOT kaput megvalósító komponens	43
<a href="#">OR.h</a>	
OR kaput megvalósító komponens	45
<a href="#">PRINT.h</a>	
A kábel értéket kimenetre író komponens	46
<a href="#">STD_INP.h</a>	
Olyan INP komponens, aminek az értékét a standard bemeneten lehet megadni	48
<a href="#">XOR.h</a>	
XOR kaput megvalósító komponens	50
<a href="#">component.h</a>	
A komponensek absztrakt bázisosztálya	51
<a href="#">logic_network.h</a>	
Logikai hálózatot megvalósító osztály. Eltárolja a neki átadott komponenseket, használat után törli azokat	52
<a href="#">logic_network_configurer.h</a>	
Fájlba menthető és fájlból beolvasható logikai hálózat osztály	54
<a href="#">main.cpp</a>	
Main file az összes teszttel Minden teszt gtest_lita-al írt. Az utolsó tesztet standard bemenetről olvas	56
<a href="#">wire.h</a>	
Vezetéket megvalósító osztály	56



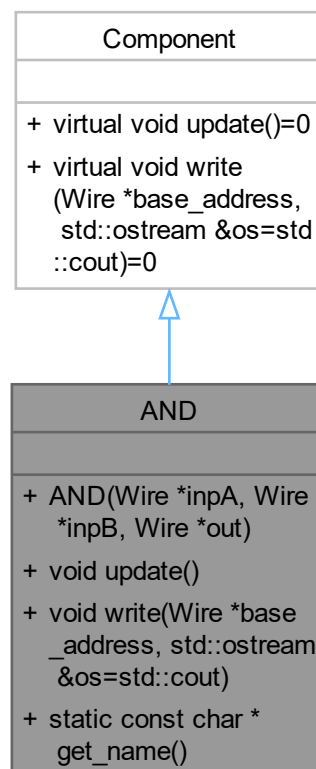


## 5. fejezet

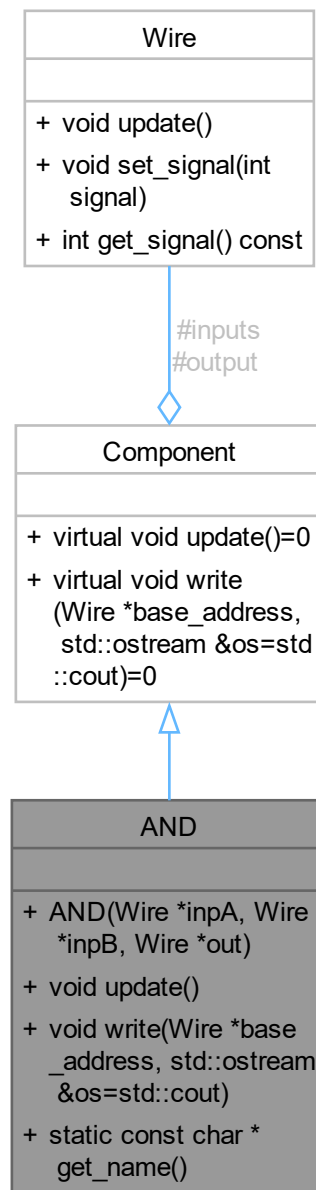
# Osztályok dokumentációja

### 5.1. AND osztályreferencia

Az AND osztály származási diagramja:



Az AND osztály együttműködési diagramja:



### Publikus tagfüggvények

- **AND (Wire \*inpA, Wire \*inpB, Wire \*out)**  
*AND komponens konstruktora.*
- **void update ()**  
*Kimeneti jel a bemeneteken végzett AND eredménye.*
- **void write (Wire \*base\_address, std::ostream &os=std::cout)**  
*Komponens kiírása a fájlba mentéshez.*

**Statikus publikus tagfüggvények**

- static const char \* [get\\_name](#) ()

Visszaadja a komponens elmentésekor használt nevet.

**5.1.1. Konstruktorkok és destruktorkok dokumentációja****5.1.1.1. AND()**

```
AND::AND (
    Wire * inpA,
    Wire * inpB,
    Wire * out)
```

[AND](#) komponens konstruktora.

**Paraméterek**

<i>inpA</i>	Első bemenet
<i>inpB</i>	Második bemenet
<i>out</i>	Kimenet

**5.1.2. Tagfüggvények dokumentációja****5.1.2.1. get\_name()**

```
static const char * AND::get_name () [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

**Visszatérési érték**

A komponens neve

**5.1.2.2. write()**

```
void AND::write (
    Wire * base_address,
    std::ostream & os = std::cout) [virtual]
```

Komponens kiírása a fájlba mentéshez.

**Paraméterek**

<i>base_address</i>	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
<i>os</i>	A kimeneti adatfolyam, amire a kapu kiírja magát

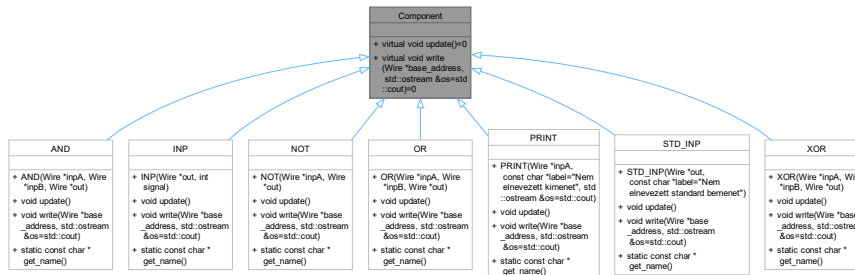
Megvalósítja a következőket: [Component](#).

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

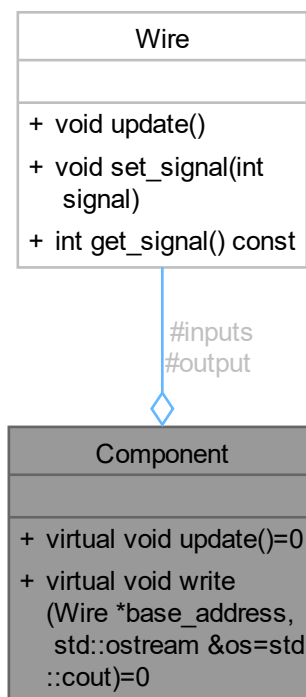
- AND.h
- AND.cpp

## 5.2. Component osztályreferencia

A Component osztály származási diagramja:



A Component osztály együttműködési diagramja:



### Publikus tagfüggvények

- virtual void **update** ()=0

*Kaputól függően a frissítés elvégzése:*

*Bementi jelekből a kimeneti jelek kiszámítása / egyéb műveletek végrehajtása.*

- virtual void **write** (Wire \*base\_address, std::ostream &os=std::cout)=0

*Komponens kiírása a fájlba mentéshez.*

## 5.2.1. Tagfüggvények dokumentációja

### 5.2.1.1. write()

```
virtual void Component::write (
    Wire * base_address,
    std::ostream & os = std::cout) [pure virtual]
```

Komponens kiírása a fájlba mentéshez.

#### Paraméterek

<i>base_address</i>	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
<i>os</i>	A kimeneti adatfolyam, amire a kapu kiírja magát

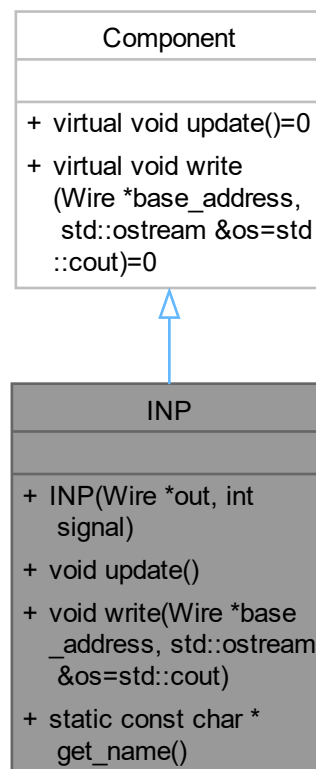
Megvalósítják a következők: [AND](#), [INP](#), [NOT](#), [OR](#), [PRINT](#), [STD\\_INP](#) és [XOR](#).

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

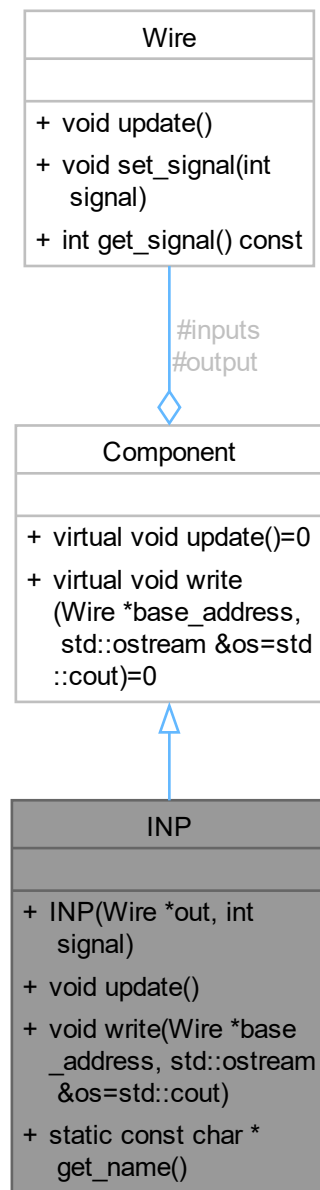
- [component.h](#)
- [component.cpp](#)

## 5.3. INP osztályreferencia

Az INP osztály származási diagramja:



Az INP osztály együttműködési diagramja:



### Publikus tagfüggvények

- **INP** (**Wire** \*out, int signal)  
*INP komponens konstruktora.*
- void **update** ()  
*A belső jelet kirakja a kimeneti kábelre.*
- void **write** (**Wire** \*base\_address, std::ostream &os=std::cout)  
*Komponens kiírása a fájlba mentéshez.*

### Statikus publikus tagfüggvények

- static const char \* [get\\_name](#) ()  
Visszaadja a komponens elmentésekor használt nevet.

## 5.3.1. Konstruktorok és destruktorok dokumentációja

### 5.3.1.1. INP()

```
INP::INP (
    Wire * out,
    int signal)
```

[INP](#) komponens konstruktora.

#### Paraméterek

<i>out</i>	Kimenet
------------	---------

## 5.3.2. Tagfüggvények dokumentációja

### 5.3.2.1. get\_name()

```
static const char * INP::get_name () [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

#### Visszatérési érték

A komponens neve

### 5.3.2.2. write()

```
void INP::write (
    Wire * base_address,
    std::ostream & os = std::cout) [virtual]
```

Komponens kiírása a fájlba mentéshez.

#### Paraméterek

<i>base_address</i>	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
<i>os</i>	A kimeneti adatfolyam, amire a kapu kiírja magát

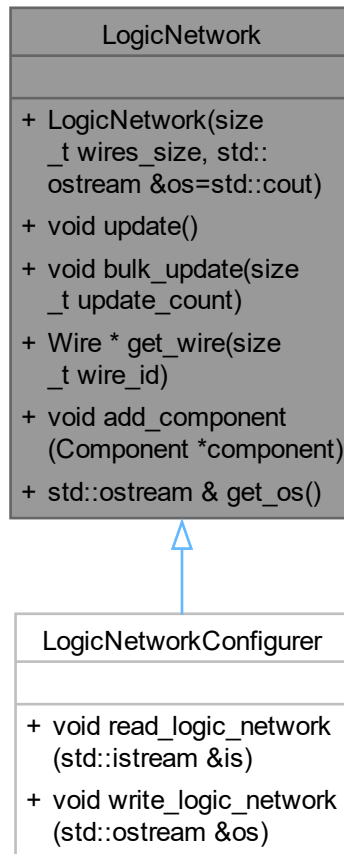
Megvalósítja a következőket: [Component](#).

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- [INP.h](#)
- [INP.cpp](#)

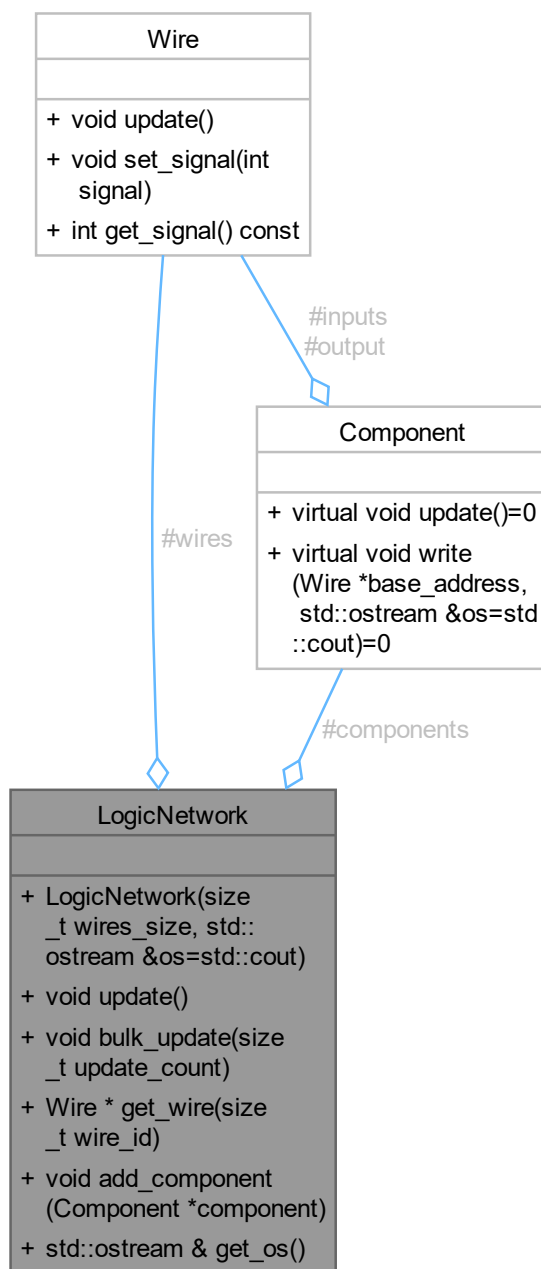
## 5.4. LogicNetwork osztályreferencia

A LogicNetwork osztály származási diagramja:





A LogicNetwork osztály együttműködési diagramja:



#### Publikus tagfüggvények

- **LogicNetwork** (size\_t wires\_size, std::ostream &os=std::cout)  
*Logikai hálózat konstruktora.*
- void **update** ()  
*A hálózat elemeinek frissítése: először kábelek, majd komponensek sorrendben.*
- void **bulk\_update** (size\_t update\_count)

Egyszerre több frissítés futtatása. Kiírja az egyes frissítések kezdetét az adatfolyamra.

- `Wire * get_wire (size_t wire_id)`

Visszaad egy mutatót az eltárolt vezetékekre sorszám alapján.

- `void add_component (Component *component)`

Hozzáad egy komponenst a hálózathoz. A komponenst a hálózat fogja felszabadítani.

- `std::ostream & get_os ()`

Visszaadja a kimeneti adatfolyamot, amire a frissítések kezdetét jelző szöveget írja ki.

## 5.4.1. Konstruktorkok és destruktorkok dokumentációja

### 5.4.1.1. LogicNetwork()

```
LogicNetwork::LogicNetwork (
    size_t wires_size,
    std::ostream & os = std::cout)
```

Logikai hálózat konstruktora.

#### Paraméterek

<code>wires_size</code>	A hálózatban lévő kábelek maximális száma
<code>os</code>	Az adatfolyam, ahova a frissítéseket elválasztó üzeneteket írja

## 5.4.2. Tagfüggvények dokumentációja

### 5.4.2.1. add\_component()

```
void LogicNetwork::add_component (
    Component * component)
```

Hozzáad egy komponenst a hálózathoz. A komponenst a hálózat fogja felszabadítani.

#### Paraméterek

<code>component</code>	Az új komponens mutatóval, amit hozzáad
------------------------	---

### 5.4.2.2. bulk\_update()

```
void LogicNetwork::bulk_update (
    size_t update_count)
```

Egyszerre több frissítés futtatása. Kiírja az egyes frissítések kezdetét az adatfolyamra.

#### Paraméterek

<code>update_count</code>	A frissítések száma
---------------------------	---------------------

### 5.4.2.3. get\_os()

```
std::ostream & LogicNetwork::get_os () [inline]
```

Visszaadja a kimeneti adatfolyamot, amire a frissítések kezdetét jelző szöveget írja ki.

#### Visszatérési érték

std::ostream&

### 5.4.2.4. get\_wire()

```
Wire * LogicNetwork::get_wire (
    size_t wire_id)
```

Visszaad egy mutatót az eltárolt vezetékekre sorszám alapján.

#### Kivételek

<i>const_char*</i>	Ha nincsen wire_id számú kábel, "Nincs elég kábel!" kivételt dob
--------------------	--

#### Paraméterek

<i>wire</i> ↔ <i>_id</i>	Vezeték sorszáma
-----------------------------	------------------

#### Visszatérési érték

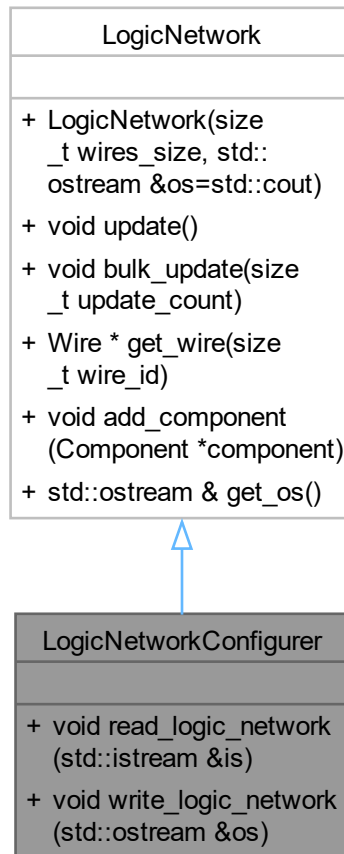
Wire\*

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

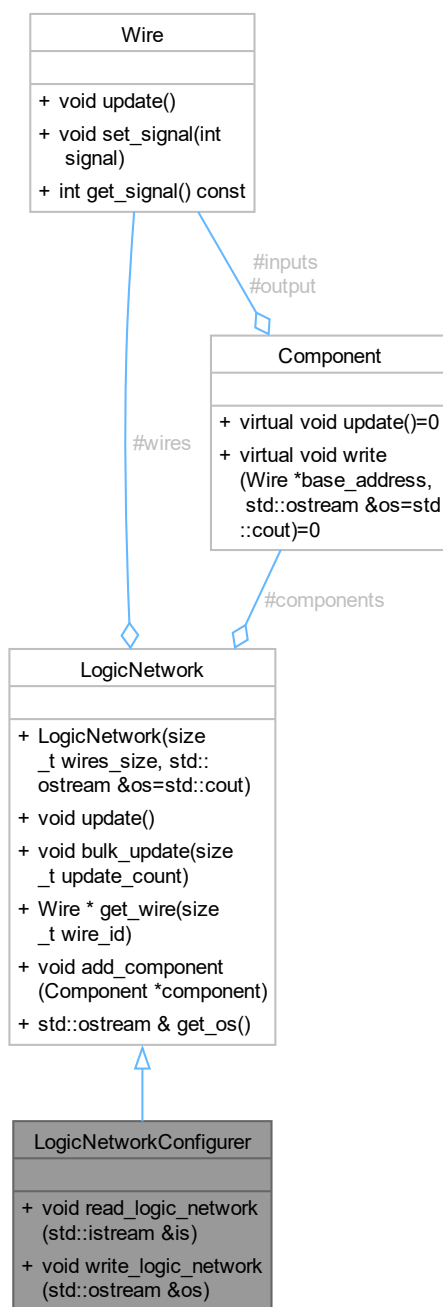
- [logic\\_network.h](#)
- [logic\\_network.cpp](#)

## 5.5. LogicNetworkConfigurer osztályreferencia

A LogicNetworkConfigurer osztály származási diagramja:



A LogicNetworkConfigurer osztály együttműködési diagramja:



### Publikus tagfüggvények

- void `read_logic_network` (std::istream &is)  
Beolvassa a kapott bemeneti adatfolyamról a hálózatot.
- void `write_logic_network` (std::ostream &os)  
Kiírja egy kimeneti adatfolyamra a hálózatot.

## Publikus tagfüggvények a(z) **LogicNetwork** osztályból származnak

- **LogicNetwork** (size\_t wires\_size, std::ostream &os=std::cout)  
*Logikai hálózat konstruktora.*
- void **update** ()  
*A hálózat elemeinek frissítése: először kábelek, majd komponensek sorrendben.*
- void **bulk\_update** (size\_t update\_count)  
*Egyszerre több frissítés futtatása. Kiírja az egyes frissítések kezdetét az adatfolyamra.*
- **Wire** \* **get\_wire** (size\_t wire\_id)  
*Visszaad egy mutatót az eltárolt vezetékekre sorszám alapján.*
- void **add\_component** (**Component** \*component)  
*Hozzáad egy komponenst a hálózathoz. A komponenst a hálózat fogja felszabadítani.*
- std::ostream & **get\_os** ()  
*Visszaadja a kimeneti adatfolyamot, amire a frissítések kezdetét jelző szöveget írja ki.*

### 5.5.1. Tagfüggvények dokumentációja

#### 5.5.1.1. read\_logic\_network()

```
void LogicNetworkConfigurer::read_logic_network (
    std::istream & is)
```

Beolvassa a kapott bemeneti adatfolyamról a hálózatot.

##### Kivételek

<code>const_char*</code>	A különböző hibaeseteket leíró cstringet dob, ha nem sikerült a beolvasás.
--------------------------	--

##### Paraméterek

<code>is</code>	A bemeneti adatfolyam
-----------------	-----------------------

#### 5.5.1.2. write\_logic\_network()

```
void LogicNetworkConfigurer::write_logic_network (
    std::ostream & os)
```

Kiírja egy kimeneti adatfolyamra a hálózatot.

##### Paraméterek

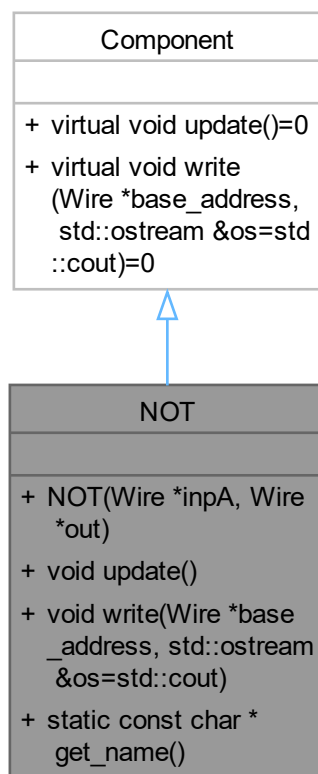
<code>os</code>	A kimeneti adatfolyam
-----------------	-----------------------

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

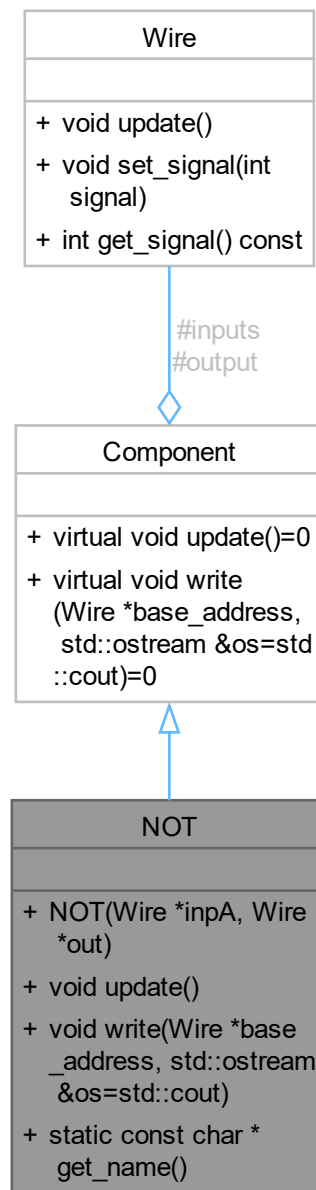
- `logic_network_configurer.h`
- `logic_network_configurer.cpp`

## 5.6. NOT osztályreferencia

A NOT osztály származási diagramja:



A NOT osztály együttműködési diagramja:



### Publikus tagfüggvények

- `NOT (Wire *inpA, Wire *out)`  
*NOT komponens konstruktora.*
- `void update ()`  
*Kimeneti jel a bemeneten végzett NOT eredménye.*
- `void write (Wire *base_address, std::ostream &os=std::cout)`  
*Komponens kiírása a fájlba mentéshez.*



### Statikus publikus tagfüggvények

- static const char \* [get\\_name](#) ()  
Visszaadja a komponens elmentésekor használt nevet.

## 5.6.1. Konstruktorok és destruktorok dokumentációja

### 5.6.1.1. NOT()

```
NOT::NOT (
    Wire * inpA,
    Wire * out)
```

[NOT](#) komponens konstruktora.

#### Paraméterek

<i>inpA</i>	Bemenet
<i>out</i>	Kimenet

## 5.6.2. Tagfüggvények dokumentációja

### 5.6.2.1. get\_name()

```
static const char * NOT::get_name () [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

#### Visszatérési érték

A komponens neve

### 5.6.2.2. write()

```
void NOT::write (
    Wire * base_address,
    std::ostream & os = std::cout) [virtual]
```

Komponens kiírása a fájlba mentéshez.

#### Paraméterek

<i>base_address</i>	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
<i>os</i>	A kimeneti adatfolyam, amire a kapu kiírja magát

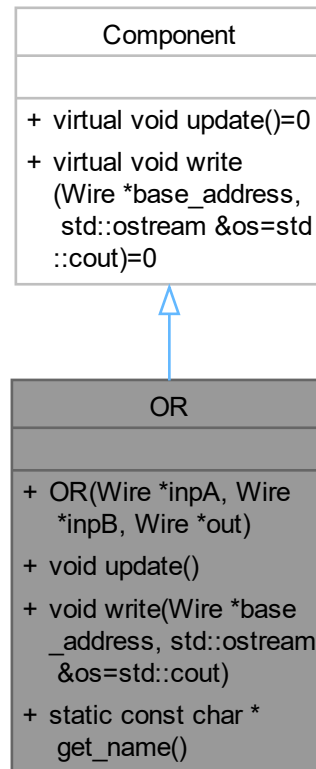
Megvalósítja a következőket: [Component](#).

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

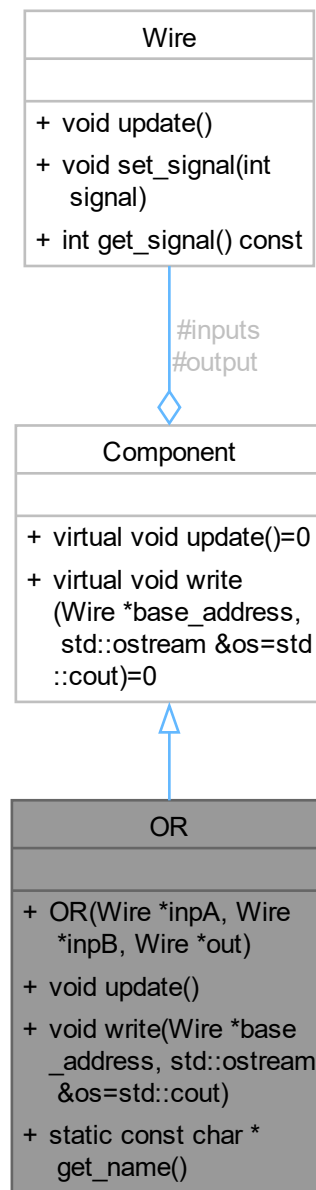
- [NOT.h](#)
- [NOT.cpp](#)

## 5.7. OR osztályreferencia

Az OR osztály származási diagramja:



Az OR osztály együttműködési diagramja:



### Publikus tagfüggvények

- **OR** (**Wire** \*inpA, **Wire** \*inpB, **Wire** \*out)  
*OR komponens konstruktora.*
- void **update** ()  
*Kimeneti jel a bemeneteken végzett OR eredménye.*
- void **write** (**Wire** \*base\_address, std::ostream &os=std::cout)  
*Komponens kiírása a fájlba mentéshez.*

## Statikus publikus tagfüggvények

- static const char \* [get\\_name](#) ()

Visszaadja a komponens elmentésekor használt nevet.

## 5.7.1. Konstruktorok és destruktorok dokumentációja

### 5.7.1.1. OR()

```
OR::OR (
    Wire * inpA,
    Wire * inpB,
    Wire * out)
```

[OR](#) komponens konstruktora.

#### Paraméterek

<i>inpA</i>	Első bemenet
<i>inpB</i>	Második bemenet
<i>out</i>	Kimenet

## 5.7.2. Tagfüggvények dokumentációja

### 5.7.2.1. get\_name()

```
static const char * OR::get_name () [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

#### Visszatérési érték

A komponens neve

### 5.7.2.2. write()

```
void OR::write (
    Wire * base_address,
    std::ostream & os = std::cout) [virtual]
```

Komponens kiírása a fájlba mentéshez.

#### Paraméterek

<i>base_address</i>	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
<i>os</i>	A kimeneti adatfolyam, amire a kapu kiírja magát

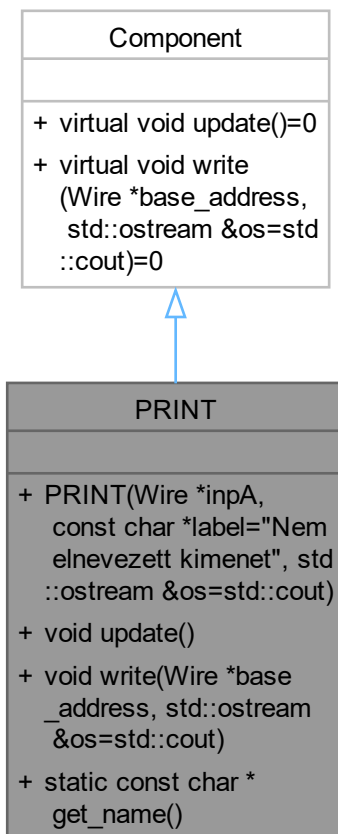
Megvalósítja a következőket: [Component](#).

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

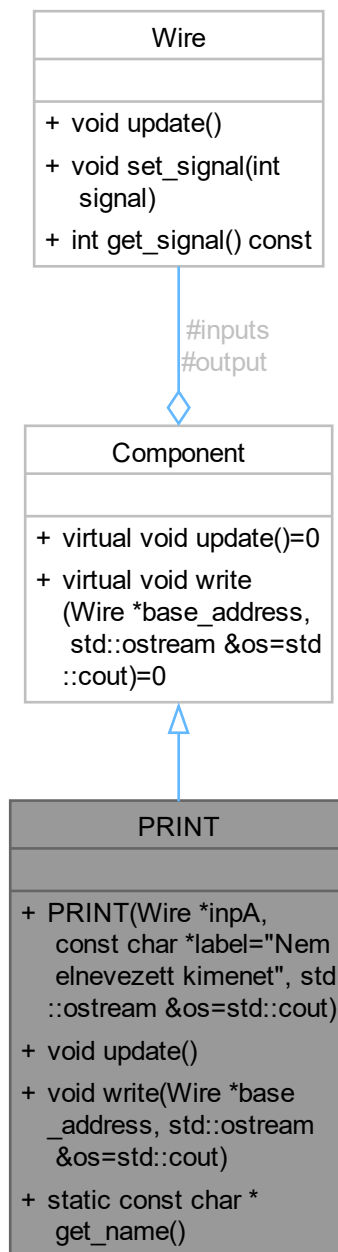
- [OR.h](#)
- [OR.cpp](#)

## 5.8. PRINT osztályreferencia

A PRINT osztály származási diagramja:



A PRINT osztály együttműködési diagramja:



### Publikus tagfüggvények

- **PRINT** (`Wire *inpA`, `const char *label="Nem elnevezett kimenet"`, `std::ostream &os=std::cout`)  
*PRINT komponens konstruktora.*
- `void update ()`  
*Kiírja a bemeneti kábel értékét a megadott adatfolyamra.*
- `void write (Wire *base_address, std::ostream &os=std::cout)`  
*Komponens kiírása a fájlba mentéshez.*

### Statikus publikus tagfüggvények

- static const char \* [get\\_name](#) ()  
Visszaadja a komponens elmentésekor használt nevet.

## 5.8.1. Konstruktorkok és destruktorok dokumentációja

### 5.8.1.1. PRINT()

```
PRINT::PRINT (
    Wire * inpA,
    const char * label = "Nem elnevezett kimenet",
    std::ostream & os = std::cout)
```

[PRINT](#) komponens konstruktora.

#### Paraméterek

<i>inpA</i>	Bemeneti kábel
<i>label</i>	A címke, ami alapján meg lehet különböztetni a kimeneteket
<i>os</i>	Az adatfolyam, amire a kimenetét írja

## 5.8.2. Tagfüggvények dokumentációja

### 5.8.2.1. get\_name()

```
static const char * PRINT::get_name () [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

#### Visszatérési érték

A komponens neve

### 5.8.2.2. write()

```
void PRINT::write (
    Wire * base_address,
    std::ostream & os = std::cout) [virtual]
```

Komponens kiírása a fájlba mentéshez.

#### Paraméterek

<i>base_address</i>	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
<i>os</i>	A kimeneti adatfolyam, amire a kapu kiírja magát

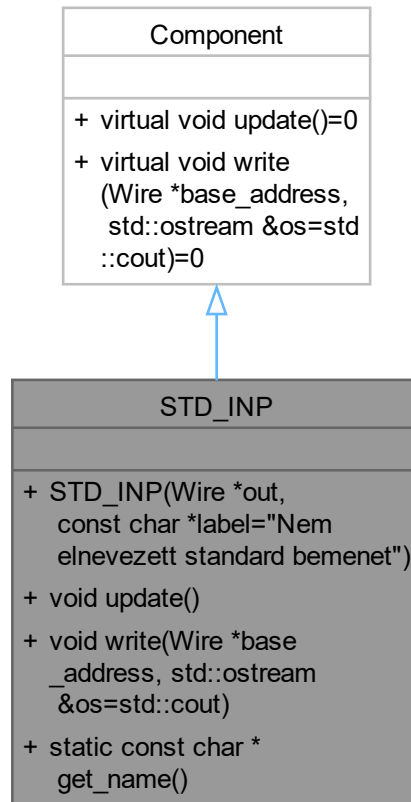
Megvalósítja a következőket: [Component](#).

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- [PRINT.h](#)
- [PRINT.cpp](#)

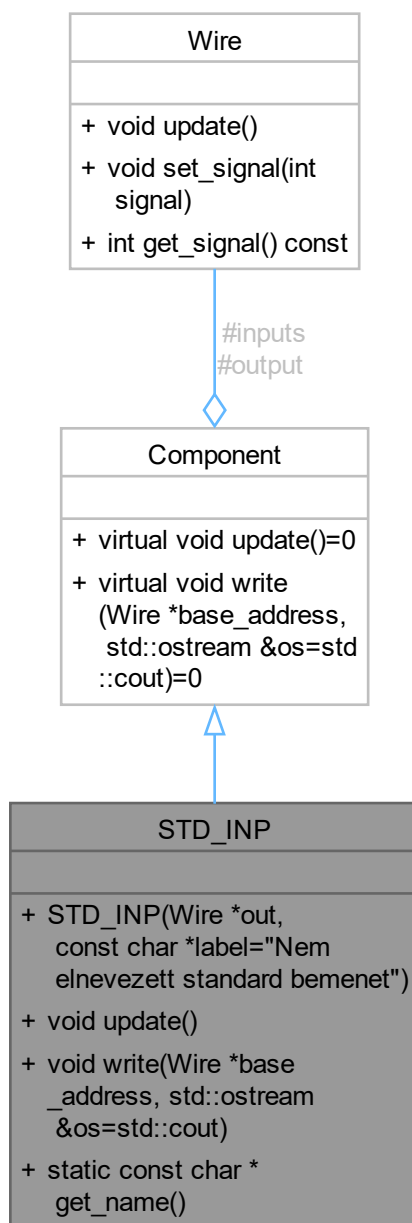
## 5.9. STD\_INP osztályreferencia

A STD\_INP osztály származási diagramja:





A STD\_INP osztály együttműködési diagramja:



#### Publikus tagfüggvények

- **STD\_INP** (**Wire** \*out, const char \*label="Nem elnevezett standard bemenet")  
*STD\_INP komponens konstruktora.*
- void **update** ()  
*Első frissítéskor elkéri, hogy mit rakjon a kimenetre. Ez után ilyen értékű **INP** kapuként működik.*
- void **write** (**Wire** \*base\_address, std::ostream &os=std::cout)  
*Komponens kiírása a fájlba mentéshez.*

## Statikus publikus tagfüggvények

- static const char \* [get\\_name](#) ()

Visszaadja a komponens elmentésekor használt nevet.

## 5.9.1. Konstruktorkok és destruktorkok dokumentációja

### 5.9.1.1. STD\_INP()

```
STD_INP::STD_INP (
    Wire * out,
    const char * label = "Nem elnevezett standard bemenet")
```

[STD\\_INP](#) komponens konstruktora.

#### Paraméterek

<i>out</i>	Kimenet
<i>label</i>	A bemenetek megkülönböztetésére használt címke

## 5.9.2. Tagfüggvények dokumentációja

### 5.9.2.1. get\_name()

```
static const char * STD_INP::get_name () [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

#### Visszatérési érték

A komponens neve

### 5.9.2.2. update()

```
void STD_INP::update () [virtual]
```

Első frissítéskor elkéri, hogy mit rakjon a kimenetre. Ez után ilyen értékű [INP](#) kapuként működik.

#### Kivételek

<i>const_char*</i>	Első frissítésnél "Hibás bemenet egy STD_INP kapunak" kivételt dob, ha a kapott érték nem 0 vagy 1
--------------------	--

Megvalósítja a következőket: [Component](#).

### 5.9.2.3. write()

```
void STD_INP::write (
    Wire * base_address,
    std::ostream & os = std::cout) [virtual]
```

Komponens kiírása a fájlba mentéshez.

## Paraméterek

<i>base_address</i>	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
<i>os</i>	A kimeneti adatfolyam, amire a kapu kiírja magát

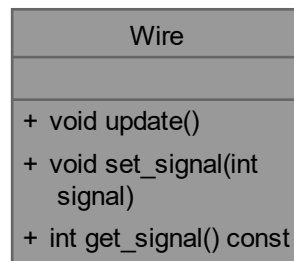
Megvalósítja a következőket: [Component](#).

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- [STD\\_INP.h](#)
- [STD\\_INP.cpp](#)

## 5.10. Wire osztályreferencia

A Wire osztály együttműködési diagramja:



### Publikus tagfüggvények

- void **update** ()  
*A vezetékek elejétől a végére rakja a jelet, elejét 0-ra állítja.*
- void [set\\_signal](#) (int signal)  
*Beállítja a bemenetén a jelet, a kapott és a már ott lévő érték közül a nagyobbra.*
- int [get\\_signal](#) () const  
*Visszaadja a kimenetén lévő jelet.*

### 5.10.1. Tagfüggvények dokumentációja

#### 5.10.1.1. get\_signal()

```
int Wire::get_signal () const [inline]
```

Visszaadja a kimenetén lévő jelet.

Visszatérési érték

int

### 5.10.1.2. set\_signal()

```
void Wire::set_signal (
    int signal)
```

Beállítja a bemenetén a jelet, a kapott és a már ott lévő érték közül a nagyobbra.

#### Paraméterek

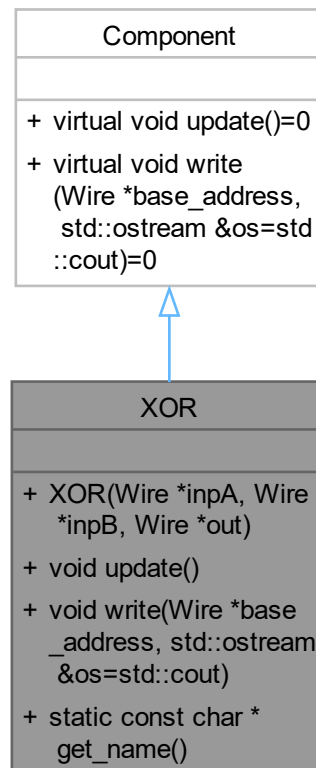
<i>signal</i>	
---------------	--

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

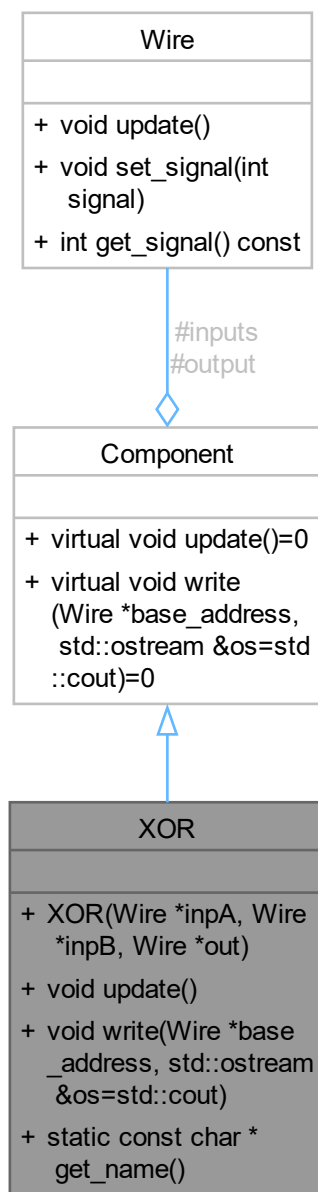
- [wire.h](#)
- [wire.cpp](#)

## 5.11. XOR osztályreferencia

A XOR osztály származási diagramja:



A XOR osztály együttműködési diagramja:



### Publikus tagfüggvények

- `XOR (Wire *inpA, Wire *inpB, Wire *out)`  
*XOR komponens konstruktora.*
- `void update ()`  
*Kimeneti jel a bemeneteken végzett XOR eredménye.*
- `void write (Wire *base_address, std::ostream &os=std::cout)`  
*Komponens kiírása a fájlba mentéshez.*

## Statikus publikus tagfüggvények

- static const char \* [get\\_name](#) ()

Visszaadja a komponens elmentésekor használt nevet.

## 5.11.1. Konstruktorok és destruktorok dokumentációja

### 5.11.1.1. XOR()

```
XOR::XOR (
    Wire * inpA,
    Wire * inpB,
    Wire * out)
```

[XOR](#) komponens konstruktora.

#### Paraméterek

<i>inpA</i>	Első bemenet
<i>inpB</i>	Második bemenet
<i>out</i>	Kimenet

## 5.11.2. Tagfüggvények dokumentációja

### 5.11.2.1. get\_name()

```
static const char * XOR::get_name () [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

#### Visszatérési érték

A komponens neve

### 5.11.2.2. write()

```
void XOR::write (
    Wire * base_address,
    std::ostream & os = std::cout) [virtual]
```

Komponens kiírása a fájlba mentéshez.

#### Paraméterek

<i>base_address</i>	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
<i>os</i>	A kimeneti adatfolyam, amire a kapu kiírja magát

Megvalósítja a következőket: [Component](#).

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- [XOR.h](#)
- XOR.cpp

## 6. fejezet

# Fájlok dokumentációja

### 6.1. basic\_components.h

```
00001 #ifndef BASIC_COMPONENTS_H
00002 #define BASIC_COMPONENTS_H
00003 // alkatalógusból importálja az egyszerű komponenseket
00004 #include "basic_components/AND.h"
00005 #include "basic_components/INP.h"
00006 #include "basic_components/NOT.h"
00007 #include "basic_components/OR.h"
00008 #include "basic_components/PRINT.h"
00009 #include "basic_components/STD_INP.h"
00010 #include "basic_components/XOR.h"
00011 #endif
```

### 6.2. AND.h

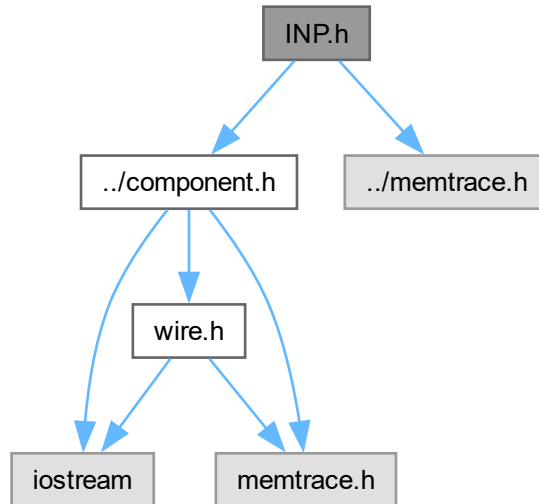
```
00001
00002 #ifndef AND_H
00003 #define AND_H
00004
00005 #include "../component.h"
00006 #include "../memtrace.h"
00007
00008 class AND : public Component {
00009     static const char* name;
00010     // copy constructor és értékadó operátor letiltása
00011     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponens létrehozni
00012     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00013     AND(const AND&);
00014     AND& operator=(const AND&);
00015
00016 public:
00017     AND(Wire* inpA, Wire* inpB, Wire* out);
00018     void update();
00019     void write(Wire* base_address, std::ostream& os = std::cout);
00020     static const char* get_name() {
00021         return name;
00022     }
00023 };
00024 #endif
```

### 6.3. INP.h fájlreferencia

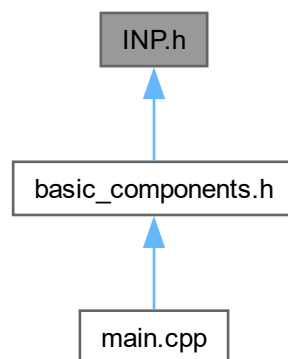
Kábelre konstans értéket író komponens.

```
#include "../component.h"  
#include "../memtrace.h"
```

Az INP.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



## Osztályok

- class [INP](#)

### 6.3.1. Részletes leírás

Kábelre konstans értéket író komponens.



## 6.4. INP.h

[Ugrás a fájl dokumentációjához.](#)

```

00001
00005 #ifndef INP_H
00006 #define INP_H
00007
00008 #include "../component.h"
00009 #include "../memtrace.h"
00010
00011 class INP : public Component {
00012     static const char* name;
00013     int signal;
00014     // copy constructor és értékadó operátor letiltása
00015     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponenst létrehozni
00016     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00017     INP(const INP&);
00018     INP& operator=(const INP&);
00019
00020 public:
00026     INP(Wire* out, int signal);
00030     void update();
00031     void write(Wire* base_address, std::ostream& os = std::cout);
00037     static const char* get_name() {
00038         return name;
00039     }
00040 };
00041
00042 #endif

```

## 6.5. NOT.h fájlreferencia

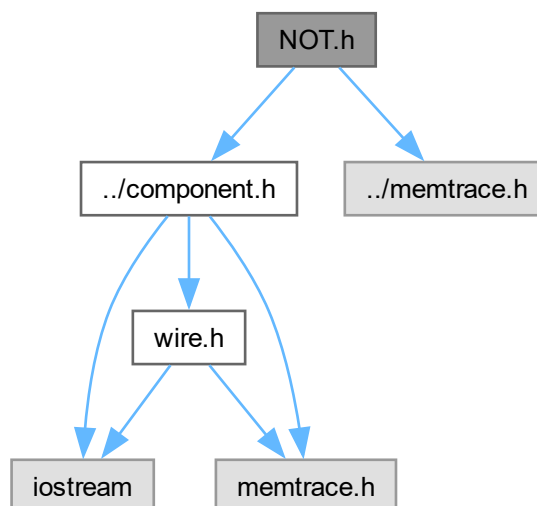
**NOT** kaput megvalósító komponens.

```

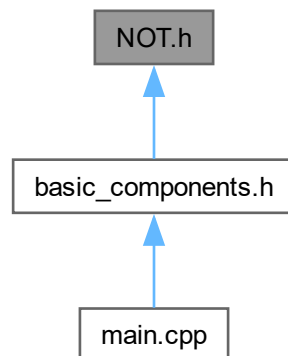
#include "../component.h"
#include "../memtrace.h"

```

A NOT.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



## Osztályok

- class [NOT](#)

### 6.5.1. Részletes leírás

[NOT](#) kaput megvalósító komponens.

## 6.6. NOT.h

[Ugrás a fájl dokumentációjához.](#)

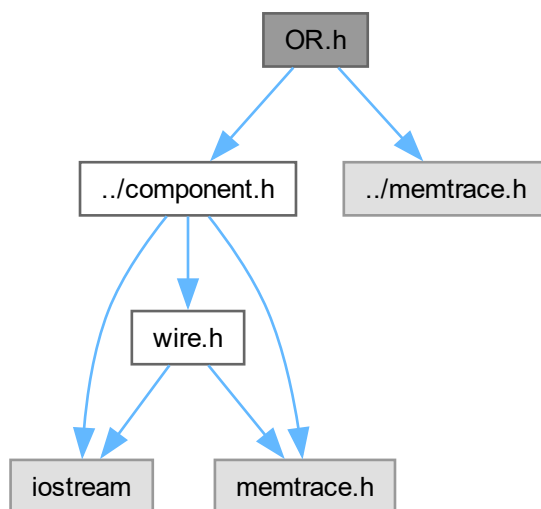
```
00001
00005 #ifndef NOT_H
00006 #define NOT_H
00007
00008 #include "../component.h"
00009 #include "../memtrace.h"
00010
00011 class NOT : public Component {
00012     static const char* name;
00013
00014     // copy constructor és értékadó operátor letiltása
00015     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponens létrehozni
00016     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00017     NOT(const NOT&);
00018     NOT& operator=(const NOT&);
00019
00020 public:
00027     NOT(Wire* inpA, Wire* out);
00031     void update();
00032     void write(Wire* base_address, std::ostream& os = std::cout);
00038     static const char* get_name() {
00039         return name;
00040     }
00041 };
00042
00043 #endif
```

## 6.7. OR.h fájlreferencia

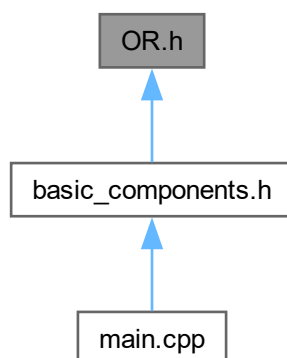
OR kaput megvalósító komponens.

```
#include "../component.h"  
#include "../memtrace.h"
```

Az OR.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



### Osztályok

- class [OR](#)

### 6.7.1. Részletes leírás

[OR](#) kaput megvalósító komponens.

## 6.8. OR.h

[Ugrás a fájl dokumentációjához.](#)

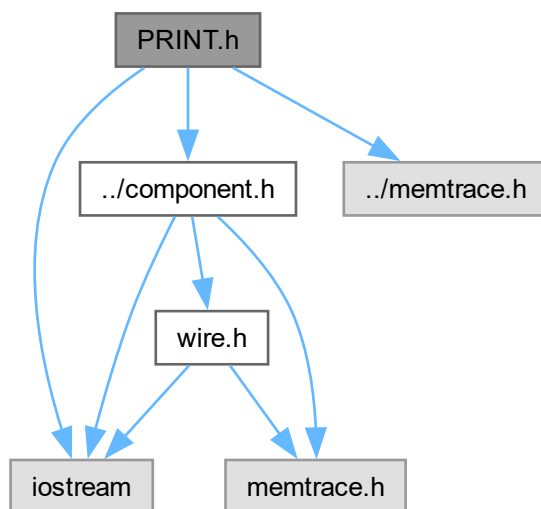
```
00001
00005
00006 #ifndef OR_H
00007 #define OR_H
00008 #include "../component.h"
00009 #include "../memtrace.h"
00010
00011 class OR : public Component {
00012     static const char* name;
00013     // copy constructor és értékadó operátor letiltása
00014     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponens létrehozni
00015     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00016     OR(const OR&);
00017     OR& operator=(const OR&);
00018
00019 public:
00027     OR(Wire* inpA, Wire* inpB, Wire* out);
00031     void update();
00032     void write(Wire* base_address, std::ostream& os = std::cout);
00038     static const char* get_name() {
00039         return name;
00040     }
00041 };
00042
00043 #endif
```

## 6.9. PRINT.h fájlreferencia

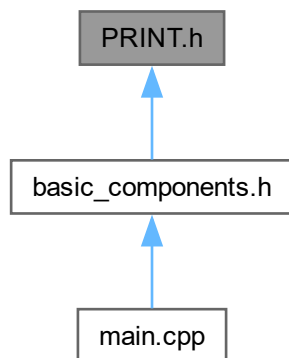
A kábel értéket kimenetre író komponens.

```
#include <iostream>
#include "../component.h"
#include "../memtrace.h"
```

A PRINT.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



## Osztályok

- class [PRINT](#)

### 6.9.1. Részletes leírás

A kábel értéket kimenetre író komponens.

## 6.10. PRINT.h

[Ugrás a fájl dokumentációjához.](#)

```

00001
00007 #ifndef PRINT_H
00008 #define PRINT_H
00009
00010 #include <iostream>
00011
00012 #include "../component.h"
00013 #include "../memtrace.h"
00014 class PRINT : public Component {
00015     static const char* name;
00016     std::ostream& os;
00017     char* label;
00018
00019     // copy constructor és értékadó operátor letiltása
00020     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponens létrehozni
00021     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00022     PRINT(const PRINT&);
00023     PRINT& operator=(const PRINT&);
00024
00025 public:
00033     PRINT(Wire* inpA, const char* label = "Nem elnevezett kimenet", std::ostream& os = std::cout);
00038     void update();
00039     void write(Wire* base_address, std::ostream& os = std::cout);
00045     static const char* get_name() {
00046         return name;
00047     }
00048     ~PRINT();
00049 };
00050
00051 #endif

```

## 6.11. STD\_INP.h fájlreferencia

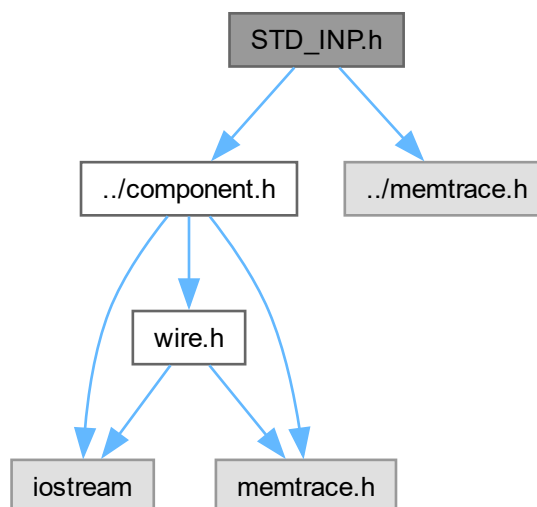
Olyan [INP](#) komponens, aminek az értékét a standard bemeneten lehet megadni.

```

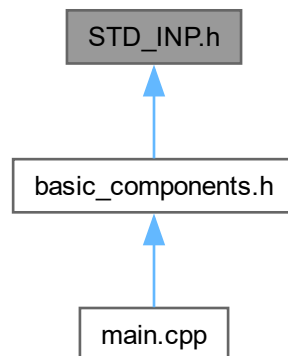
#include "../component.h"
#include "../memtrace.h"

```

A STD\_INP.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



### Osztályok

- class [STD\\_INP](#)

#### 6.11.1. Részletes leírás

Olyan [INP](#) komponens, aminek az értékét a standard bemeneten lehet megadni.

## 6.12. STD\_INP.h

[Ugrás a fájl dokumentációjához.](#)

```

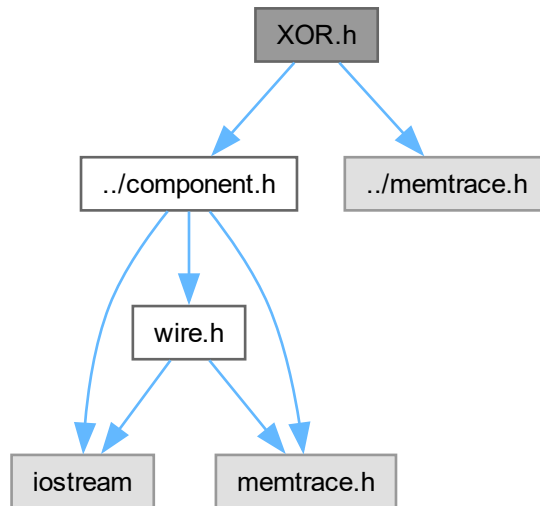
00001
00005
00006 #ifndef STD_INP_H
00007 #define STD_INP_H
00008
00009 #include "../component.h"
00010 #include "../memtrace.h"
00011
00012 // mindig a standard bemenetről olvassa be, hogy mit adjon ki
00013 class STD_INP : public Component {
00014     static const char* name;
00015     int signal;
00016     bool is_signal_set;
00017     char* label;
00018
00019     // copy constructor és értékadó operátor letiltása
00020     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponenst létrehozni
00021     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00022     STD_INP(const STD_INP&);
00023     STD_INP& operator=(const STD_INP&);
00024
00025 public:
00032     STD_INP(Wire* out, const char* label = "Nem elnevezett standard bemenet");
00037     void update();
00038     void write(Wire* base_address, std::ostream& os = std::cout);
00044     static const char* get_name() {
00045         return name;
00046     }
00047     ~STD_INP();
00048 };
00049
00050 #endif
  
```

## 6.13. XOR.h fájlreferencia

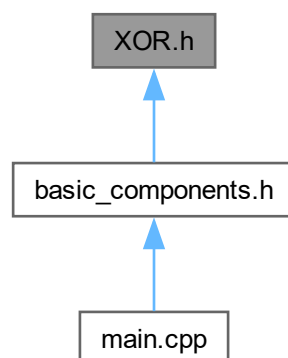
**XOR** kaput megvalósító komponens.

```
#include "../component.h"  
#include "../memtrace.h"
```

A XOR.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



### Osztályok

- class **XOR**



### 6.13.1. Részletes leírás

XOR kaput megvalósító komponens.

## 6.14. XOR.h

[Ugrás a fájl dokumentációjához.](#)

```

00001
00005 #ifndef XOR_H
00006 #define XOR_H
00007
00008 #include "../component.h"
00009 #include "../memtrace.h"
00010
00011 class XOR : public Component {
00012     static const char* name;
00013
00014     // copy constructor és értékadó operátor letiltása
00015     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponenst létrehozni
00016     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00017     XOR(const XOR&);
00018     XOR& operator=(const XOR&);
00019
00020 public:
00028     XOR(Wire* inpA, Wire* inpB, Wire* out);
00032     void update();
00033     void write(Wire* base_address, std::ostream& os = std::cout);
00039     static const char* get_name() {
00040         return name;
00041     }
00042 };
00043
00044 #endif

```

## 6.15. component.h fájlreferencia

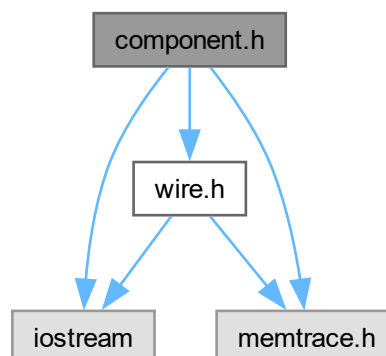
A komponensek absztrakt bázisosztálya.

```

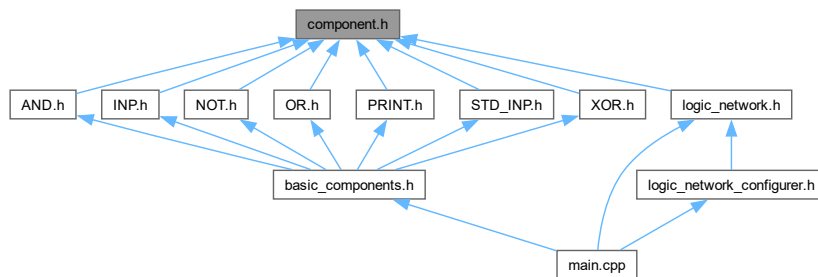
#include <iostream>
#include "memtrace.h"
#include "wire.h"

```

A component.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



## Osztályok

- class [Component](#)

### 6.15.1. Részletes leírás

A komponensek absztrakt bázisosztálya.

## 6.16. component.h

[Ugrás a fájl dokumentációjához.](#)

```

00001
00005
00006 #ifndef COMPONENT_H
00007 #define COMPONENT_H
00008 #include <iostream>
00009
00010 #include "memtrace.h"
00011 #include "wire.h"
00012
00013 class Component {
00014     // a leszámazottak állítják be, hogy mik legyenek a be és kimenetek
00015     protected:
00016         Wire** inputs;
00017         Wire* output;
00018
00019     public:
00020         Component() : inputs(nullptr), output(nullptr) {};
00025         virtual void update() = 0;
00032         virtual void write(Wire* base_address, std::ostream& os = std::cout) = 0;
00033         virtual ~Component();
00034 };
00035 #endif
  
```

## 6.17. logic\_network.h fájlreferencia

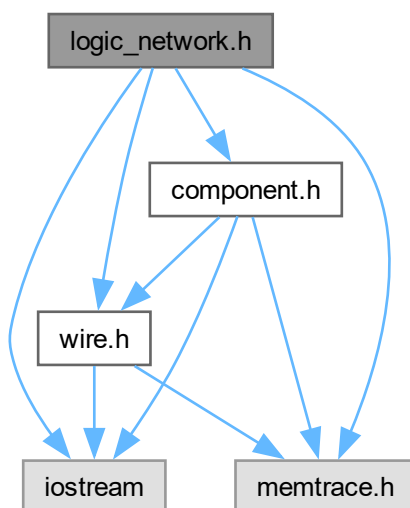
Logikai hálózatot megvalósító osztály. Eltárolja a neki átadott komponenseket, használat után törli azokat.

```

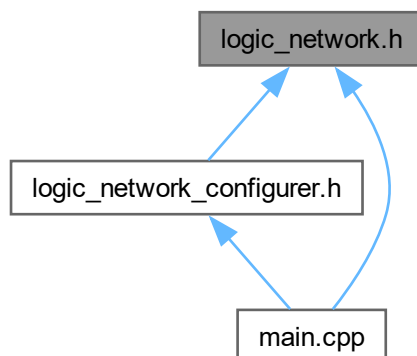
#include <iostream>
#include "component.h"
#include "memtrace.h"
  
```

```
#include "wire.h"
```

A logic\_network.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



## Osztályok

- class [LogicNetwork](#)

### 6.17.1. Részletes leírás

Logikai hálózatot megvalósító osztály. Eltárolja a neki átadott komponenseket, használat után törli azokat.

## 6.18. logic\_network.h

[Ugrás a fájl dokumentációjához.](#)

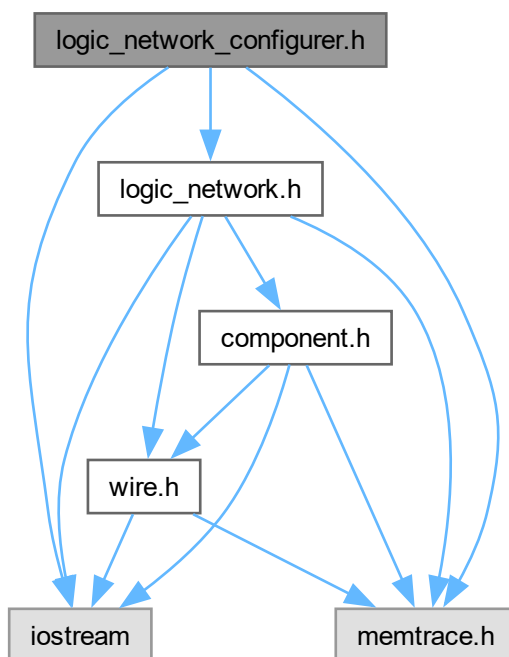
```
00001
00005 #ifndef LOGIC_NETWORK_H
00006 #define LOGIC_NETWORK_H
00007 #include <iostream>
00008
00009 #include "component.h"
00010 #include "memtrace.h"
00011 #include "wire.h"
00012 class LogicNetwork {
00013     // ne lehessen lemásolni, mert nincs sok értelme
00014     // és lehetetlen lenne lemásolni a mutatók miatt, valójában kirakná egy streamre és visszaolvasná
00015     // automatikusan a configurert se lehet lemásolni
00016     LogicNetwork(const LogicNetwork&);
00017     LogicNetwork& operator=(const LogicNetwork&);
00018
00019 protected:
00020     // UML-ben frissíteni, ez wire tömböt tárol, nem Wire*[]-t
00021     // mert asszem végül nem lehet felüldefelni a wire-t?
00022     Wire* wires;
00023     size_t wires_size;
00024
00025     Component** components;
00026     size_t components_size;
00027     // protecteden, mert nem kell tudnia a felhasználónak, hogy mennyi van a komponensekből, meg hogy
    mik azok
00028     // viszont ki szeretném írni
00029
00030     std::ostream& os;
00031
00032 public:
00033     LogicNetwork(size_t wires_size, std::ostream& os = std::cout);
00034     void update();
00035     void bulk_update(size_t update_count);
00036
00037     Wire* get_wire(size_t wire_id);
00038     void add_component(Component* component);
00039
00040     std::ostream& get_os() { return os; }
00041     virtual ~LogicNetwork();
00042 };
00043 #endif
```

## 6.19. logic\_network\_configurer.h fájlreferencia

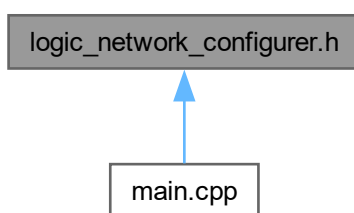
Fájlba menthető és fájlból beolvasható logikai hálózat osztály.

```
#include <iostream>
#include "logic_network.h"
#include "memtrace.h"
```

A logic\_network\_configurer.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



## Osztályok

- class [LogicNetworkConfigurer](#)

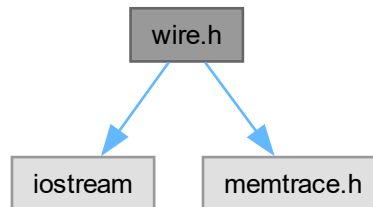
### 6.19.1. Részletes leírás

Fájlba menthető és fájlból beolvasható logikai hálózat osztály.

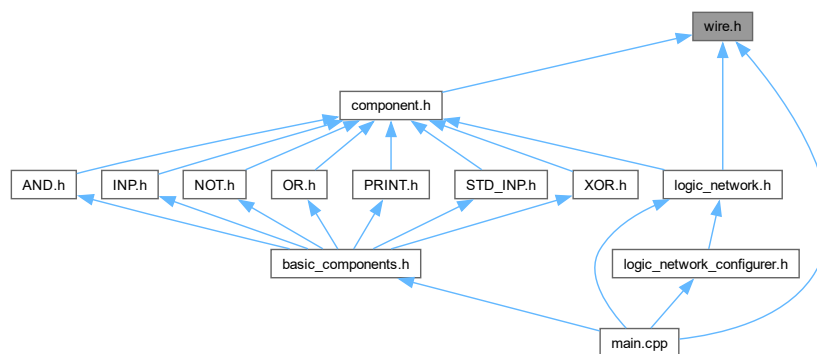


```
#include <iostream>
#include "memtrace.h"
```

A wire.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



## Osztályok

- class [Wire](#)

### 6.22.1. Részletes leírás

Vezetéket megvalósító osztály.

## 6.23. wire.h

[Ugrás a fájl dokumentációjához.](#)

```
00001
00005 #ifndef WIRE_H
00006 #define WIRE_H
00007 #include <iostream>
```

```
00008
00009 #include "memtrace.h"
00010 class Wire {
00011     int input;
00012     int output;
00013     Wire(const Wire&);
00014     Wire& operator=(const Wire&);
00015
00016 public:
00017     Wire() : input(0), output(0) {}
00022     void update();
00028     void set_signal(int signal);
00034     int get_signal() const {
00035         return output;
00036     }
00037 };
00038
00039 #endif
```



# Tárgymutató

- add\_component
  - LogicNetwork, [20](#)
- AND, [11](#)
  - AND, [13](#)
  - get\_name, [13](#)
  - write, [13](#)
- AND.h, [41](#)
- bulk\_update
  - LogicNetwork, [20](#)
- Component, [14](#)
  - write, [15](#)
- component.h, [51](#)
- Felhasználói dokumentáció, [1](#)
- get\_name
  - AND, [13](#)
  - INP, [17](#)
  - NOT, [27](#)
  - OR, [30](#)
  - PRINT, [33](#)
  - STD\_INP, [36](#)
  - XOR, [40](#)
- get\_os
  - LogicNetwork, [20](#)
- get\_signal
  - Wire, [37](#)
- get\_wire
  - LogicNetwork, [21](#)
- INP, [15](#)
  - get\_name, [17](#)
  - INP, [17](#)
  - write, [17](#)
- INP.h, [41](#), [43](#)
- logic\_network.h, [52](#)
- logic\_network\_configurer.h, [54](#)
- LogicNetwork, [18](#)
  - add\_component, [20](#)
  - bulk\_update, [20](#)
  - get\_os, [20](#)
  - get\_wire, [21](#)
  - LogicNetwork, [20](#)
- LogicNetworkConfigurer, [22](#)
  - read\_logic\_network, [24](#)
  - write\_logic\_network, [24](#)
- main.cpp, [56](#)
- NOT, [25](#)
  - get\_name, [27](#)
  - NOT, [27](#)
  - write, [27](#)
- NOT.h, [43](#), [44](#)
- OR, [28](#)
  - get\_name, [30](#)
  - OR, [30](#)
  - write, [30](#)
- OR.h, [45](#), [46](#)
- PRINT, [31](#)
  - get\_name, [33](#)
  - PRINT, [33](#)
  - write, [33](#)
- PRINT.h, [46](#), [48](#)
- read\_logic\_network
  - LogicNetworkConfigurer, [24](#)
- set\_signal
  - Wire, [37](#)
- STD\_INP, [34](#)
  - get\_name, [36](#)
  - STD\_INP, [36](#)
  - update, [36](#)
  - write, [36](#)
- STD\_INP.h, [48](#), [49](#)
- update
  - STD\_INP, [36](#)
- Wire, [37](#)
  - get\_signal, [37](#)
  - set\_signal, [37](#)
- wire.h, [56](#)
- write
  - AND, [13](#)
  - Component, [15](#)
  - INP, [17](#)
  - NOT, [27](#)
  - OR, [30](#)
  - PRINT, [33](#)
  - STD\_INP, [36](#)
  - XOR, [40](#)
- write\_logic\_network
  - LogicNetworkConfigurer, [24](#)
- XOR, [38](#)

get\_name, [40](#)  
write, [40](#)  
XOR, [40](#)  
XOR.h, [50](#), [51](#)