# Table Of Contents

# 1 System Requirements

The driver requires a uniprocessor i386 architecture machine running Microsoft Windows NT 3.51. It might also run on multiprocessor machines, but this has not been verified.

# 2 Extent of Supply

| File | Explanation |
|---|---|
| `b004.sys` | The driver executable |
| `b004.ini` | Registry default values |
| `link.h` | C Header file containing definitions needed to compile programs accessing the driver |
| `b004.wpd` | This documentation |
| `regini.exe` | This program sets up the registry using the data supplied in the file `b004.ini` |

# 3 Driver Installation

To install the new driver on a computer running Windows NT administrator rights are required. Perform these steps:

- run the command

  ```
  regini b004.ini
  ```

  from a command prompt. This sets up new registry values needed by NT and the driver.

- do

  ```
  copy b004.sys %SystemRoot%\system32\drivers
  ```

  This copies the driver executable to the Windows directory.

- reboot the machine.

# 4 Driver Deinstallation

To remove this driver from the system, you need to remove the driver executable file

```
%SystemRoot%\system32\drivers\b004.sys
```

You also need to remove all registry entries that belong to this driver. Remove

```
HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/b004
```

and

```
HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/
                                    Services/EventLog/System/b004
```

using the Windows NT program `regedt32`. Having done this, the driver is completely removed from the system.

# 5 Customizing the Driver

The driver reads several values from registry on startup determining its behaviour at runtime. Some values may be changed through ioctl functions at runtime, others may not. The installation procedure sets default values for all registry values. If you do not like these defaults, you may change them using the Windows NT
program `regedt32`. This progam is part of Windows NT, just type

```
regedt32
```

The values can be found under

```
HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/
                            Services/b004/Parameters/Devicen/
```

with *n* being the device number.The B004 driver uses these values:

| Name | Meaning |
|---|---|
| `Timeout` | This parameter specifies the timeout in milliseconds for read and write operations. It defaults to 5 seconds. |
| `BaseAddress` | This parameters specifies the port base address used for this device. It defaults to `0x150.` |

| | |
|---|---|
| `Interrupt` | This parameter specifies the interrupt used for this device. Set it to `0` if you do not want interrupts to be used. The driver then polls the device which causes more system load and degrades response time. The default interrupt is 3. |
| `PollRetry` | Transferring every single byte via interrupt would result in very poor performance. So this parameter specifies how many times the device shall be polled before waiting for the next character by interrupt. The default is 100. You should normally not need to change this value. |
| `ProcessHeader` | Specify 1 if you want the device to operate in header mode instead of stream mode by default. The default is 0. See the chapter on header mode more information. |

If you add another B004 device to your machine, simply create another registry key

```
HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/
                          Services/b004/Parameters/Devicen/
```

with *n* being the next free device number and enter all the values given above that differ from the default.

Note that the driver does neither check for all devices having unique base addresses nor interrupts. It stops checking for devices after the first missing `Devicen` tree, i.e. if you have the trees `Device1`, `Device3` and `Device4` the driver will only recognize one device.

Any changes you make in the registry will not take effect before the driver is started next time.

# 6 Starting and Stopping the Driver

Once the driver is installed on your system, it must also be started before you can use it. You can do this either by typing

```
net start b004
```

or by using the `Control Panel/Devices`. Select `b004` and click the `Start` button. Note that the driver will refuse to start if it encounters any problems. See chapter “Troubleshooting” for more information.
During the boot process, Windows NT can start drivers automatically. In order to have the B004 driver started automatically, you need to change its startup type. This is done using the `Control Panel/Devices`. Select `b004` and click the `Startup` button. Then select `Automatic` as the desired startup type.

It is not recommended to change the startup type to `Automatic` before having verified that the driver starts cleanly by starting the driver manually and checking whether Windows NTs event viewer indicates any driver related problems.
Unless the startup type is set to `Automatic`, you have to start the driver manually each time the machine is rebooted.
The driver can be stopped either by typing

```
net stop b004
```

or by using the `Control Panel/Devices`. Select `b004` and click the `Stop` button.
Note that it is not necessary to stop the driver before shutting down the machine.

# 7 Writing Your Own Programs Using the Driver

## 7.1 Introduction

Windows NT supports a fairly generic interface to allow communication between user programs and device drivers. It uses the same function calls that are used to do file I/O. Devices are separated from common files by reserving a separate name space for them. A device name has the form \\.*NameIndex*, with *Name* identifying the driver and *Index* identifying the specific device controlled by this driver. The *Name* that has to be specified to access the B004 driver is `pclink`. *Index* is 1 for the first B004 board installed, 2 for the second, and so on. Note that the C notation for a single backslash in a string is `"\\"`, so the name of the first B004 board becomes `"\\\\.\\pclink1"`.
If your program wants to perform driver specific I/O control functions using `DeviceIoControl`, it needs to include the file `link.h`.
The B004 driver supports only half duplex operation. Any device may only be opened by a single thread, though different devices may operate simultaneously.

## 7.2 CreateFile

The `CreateFile` function opens a device. It returns a handle that can be used to access the device.

```
HANDLE CreateFile (LPCTSTR lpDeviceName,
   DWORD dwDesiredAccess,
   DWORD dwShareMode,
   LPSECURITY_ATTRIBUTES lpSecurityAttributes,
   DWORD dwCreationDistribution,
   DWORD dwFlagsAndAttributes,
   HANDLE hTemplateFile);
```

**Parameters**

- `lpDeviceName`
  Points to a null-terminated string that specifies the name of the device to open.

- `dwDesiredAccess`
  Specifies the type of access to the device. An application can obtain read access, write access, read-write access, or device query access. You can use the following flag constants to build a value for this parameter. Both `GENERIC_READ` and `GENERIC_WRITE` must be set to obtain read-write access.

| Value | Meaning |
|---|---|
| `0` | Allows an application to query device attributes without actually accessing the device |
| `GENERIC_READ` | Specifies read access to the device |
| `GENERIC_WRITE` | Specifies write access to the file |

- `dwShareMode`
  Specifies how this device can be shared. This parameter must be some combination of the following values:

| Value | Meaning |
|---|---|
| `0` | Prevents the device from being shared |
| `FILE_SHARE_READ` | Other open operations can be performed on the device for read access |
| `FILE_SHARE_WRITE` | Other open operations can be performed on the device for write access |

- `lpSecurityAttributes`
  Is only meaningful for file systems. Specify `NULL` when opening devices.

- `dwCreationDistribution`
  You must specify `OPEN_EXISTING` when opening devices.

- `dwFlagsAndAttributes`
  You must specify `0` when opening devices.

- `hTemplateFile`
  You must specify `NULL` when opening devices.

**Return Value**

If the function succeeds, the return value is an open handle to the specified device. If the function fails, the return value is `INVALID_HANDLE_VALUE`. To get extended error information, call `GetLastError`.

## 7.3 ReadFile

The `ReadFile` function reads data from a device. The device handle must have been created with `GENERIC_READ` access to the device.

```
BOOL ReadFile (HANDLE hDevice,
   LPVOID lpBuffer,
   DWORD nNumberOfBytesToRead,
   LPDWORD lpNumberOfBytesRead,
   LPOVERLAPPED lpOverlapped);
```

**Parameters**

- `hDevice`
  Identifies the device to be read. Call the `CreateFile` function to obtain a device handle.

- `lpBuffer`
  Points to the buffer that receives the data read from the device.

- `nNumberOfBytesToRead`
  Specifies the number of bytes to be read from the device.

- `lpNumberOfBytesRead`
  Points to the number of bytes read. `ReadFile` sets this value to zero before doing any work or error checking.

- `lpOverlapped`
  You must specify `NULL` when accessing devices.

**Return Value**

If the function succeeds, the return value is `TRUE`. If the function fails, the return value is `FALSE`. To get extended error information, call `GetLastError`.

**Remarks**

Applications must not read from nor write to the input buffer that a read operation is using until the read operation completes. A premature access to the buffer may lead to corruption of the data read into that buffer.
The ReadFile function may fail and return `ERROR_INVALID_USER_BUFFER` or `ERROR_NOT_ENOUGH_MEMORY` whenever there are too many outstanding asynchronous I/O requests.

## 7.4 WriteFile

The `WriteFile` function writes data to a device. The device handle must have been created with `GENERIC_WRITE` access to the device.

```
BOOL WriteFile (HANDLE hDevice,
   LPCVOID lpBuffer,
   DWORD nNumberOfBytesToWrite,
   LPDWORD lpNumberOfBytesWritten,
   LPOVERLAPPED lpOverlapped);
```

**Parameters**

- `hDevice`
  Identifies the device to be written to. Call the `CreateFile` function to obtain a device handle.

- `lpBuffer`
  Points to the buffer containing the data to be written to the device.

- `nNumberOfBytesToWrite`
  Specifies the number of bytes to write to the device.

- `lpNumberOfBytesWritten`
  Points to the number of bytes written by this function call. `WriteFile` sets this value to zero before doing any work or error checking.

- `lpOverlapped`
  You must specify `NULL` when accessing devices.

**Return Value**

If the function succeeds, the return value is `TRUE`. If the function fails, the return value is `FALSE`. To get extended error information, call `GetLastError`.

**Remarks**

Applications must not read from nor write to the input buffer that a write operation is using until the write operation completes. A premature access to the buffer may lead to corruption of the data written to the device.
The `WriteFile` function may fail with `ERROR_INVALID_USER_BUFFER` or `ERROR_NOT_ENOUGH_MEMORY` whenever there are too many outstanding asynchronous I/O requests.

## 7.5 DeviceIoControl

The `DeviceIoControl` function sends a control code directly to a specified device driver, causing the corresponding device to perform the specified operation.

```
BOOL DeviceIoControl (HANDLE hDevice,
   DWORD dwIoControlCode,
   LPVOID lpInBuffer,
   DWORD nInBufferSize,
   LPVOID lpOutBuffer,
   DWORD nOutBufferSize,
   LPDWORD lpBytesReturned,
   LPOVERLAPPED lpOverlapped);
```

**Parameters**

- `hDevice`
  Handle to the device that is to perform the operation. Call the `CreateFile` function to obtain a device handle.

- `dwIoControlCode`
  Specifies the control code for the operation. This value identifies the specific operation to be performed and the type of device on which the operation is to be performed. Each device driver may define its own set of values.

- `lpInBuffer`
  Pointer to a buffer that contains the data required to perform the operation. This parameter can be `NULL` if the `dwIoControlCode` parameter specifies an operation that does not require input data.

- `nInBufferSize`
  Specifies the size, in bytes, of the buffer pointed to by `lpInBuffer`.

- `lpOutBuffer`
  Pointer to a buffer that receives the operation's output data. This parameter can be `NULL` if the `dwIoControlCode` parameter specifies an operation that does not produce output data.

- `nOutBufferSize`
  Specifies the size, in bytes, of the buffer pointed to by `lpOutBuffer`.

- `lpBytesReturned`
  Pointer to a variable that receives the size, in bytes, of the data stored into the buffer pointed to by `lpOutBuffer`. This parameter cannot be `NULL`. Even when an operation produces no output data, and `lpOutBuffer` can be `NULL`, the `DeviceIoControl` function makes use of the variable pointed to by `lpBytesReturned`. After such an operation, the value of the variable is without meaning.

- `lpOverlapped`
  You must specify NULL when accessing device drivers.

**Return Value**

If the function succeeds, the return value is `TRUE`. If the function fails, the return value is `FALSE`. To get extended error information, call `GetLastError`.

**Remarks**

The transputer link interface specific values for dwIoControlCode are defined in the file `link.h`, which must be included by your source code if you want to perform calls to `DeviceIoControl`. The B004 driver supports a subset of the codes defined there.

Legal values for `dwIoControlCode` are:

- `IOCTL_LINK_RESET_LINK`

  Resets the link and any transputer connected to it.

  **Parameters**

  - `lpInBuffer, nInBufferSize`
    InBuffer is not used. Specify `NULL, 0`.

  - `lpOutBuffer, nOutBufferSize`
    OutBuffer is not used. Specify `NULL, 0`.

  **Return Value**

  If the function succeeds, the return value is `TRUE`. If the function fails, the return value is `FALSE`. To get extended error information, call `GetLastError`.

- `IOCTL_LINK_ANALYSE_LINK`

  Perform an analyze reset on the link.

  **Parameters**

  - `lpInBuffer, nInBufferSize`
    InBuffer is not used. Specify `NULL, 0`.

  - `lpOutBuffer, nOutBufferSize`
    OutBuffer is not used. Specify `NULL, 0`.

  **Return Value**

  If the function succeeds, the return value is `TRUE`. If the function fails, the return value is `FALSE`. To get extended error information, call `GetLastError`.

- `IOCTL_LINK_TEST_ERROR`

  Returns the state of the links error line.

  **Parameters**

  - `lpInBuffer, nInBufferSize`
    InBuffer is not used. Specify `NULL, 0`.

  - `lpOutBuffer, nOutBufferSize`
    Points to an unsigned integer (32 bit) that will be set to `0` if there is no error and to `!=0` if there is an error.

  **Return Value**

  If the function succeeds, the return value is `TRUE`. The function fails if the caller supplied insufficient space to hold the result. The return value will then be `FALSE`. To get extended error information, call `GetLastError`.

- `IOCTL_LINK_TEST_READ`

  Check how many characters are held by the input FIFO (and thus can be read immediately).

  **Parameters**

  - `lpInBuffer, nInBufferSize`
    InBuffer is not used. Specify `NULL, 0`.

  - `lpOutBuffer, nOutBufferSize`
    Points to an unsigned integer (32 bit) that will be set to the number of characters available in the input FIFO. Since B004 boards do not contain any FIFOs, the only values that will occur are `0` and `1`.

  **Return Value**

  If the function succeeds, the return value is `TRUE`. The function fails if the caller supplied insufficient space for OutBuffer. The return value will then be `FALSE`. To get extended error information, call `GetLastError`.

- `IOCTL_LINK_TEST_WRITE`

  Check how many free locations are available in the output FIFO (equal to the number of characters that can be written without delay).

  **Parameters**

  - `lpInBuffer, nInBufferSize`
    InBuffer is not used. Specify `NULL, 0`.

  - `lpOutBuffer, nOutBufferSize`
    Points to an unsigned integer (32 bit) that will be set to the number of free locations available in the output FIFO. Since B004 boards do not contain any FIFOs, the only values that will occur are `0` and `1`.

  **Return Value**

  If the function succeeds, the return value is `TRUE`. The function fails if the caller supplied insufficient space for OutBuffer. The return value will then be `FALSE`. To get extended error information, call `GetLastError`.

- `IOCTL_LINK_SET_TIMEOUT`

  Sets the timeout for read and write operations.

  **Parameters**

  - `lpInBuffer, nInBufferSize`
    Points to an unsigned integer (32 bit) that contains the new timeout value in ms.

  - `lpOutBuffer, nOutBufferSize`
    OutBuffer is not used. Specify `NULL, 0`.

  **Return Value**

  If the function succeeds, the return value is `TRUE`. If the function fails, the return value is `FALSE`. To get extended error information, call `GetLastError`.

- `IOCTL_LINK_GET_TIMEOUT`

  Returns the current timeout for read and write operations.

  **Parameters**

  - `lpInBuffer, nInBufferSize`
    InBuffer is not used. Specify `NULL, 0`.

  - `lpOutBuffer, nOutBufferSize`
    Points to an unsigned integer (32 bit) that will be set to the current timeout value in ms.

  **Return Value**

  If the function succeeds, the return value is `TRUE`. If the function fails, the return value is `FALSE`. To get extended error information, call `GetLastError`.

- `IOCTL_LINK_SET_POLL_RETRY`

  Changes the devices value for the PollRetry parameter. Refer to the description of registry parameters for further information.

  **Parameters**

  - `lpInBuffer, nInBufferSize`
    Points to an unsigned integer (32 bit) that contains the new value.

  - `lpOutBuffer, nOutBufferSize`
    OutBuffer is not used. Specify `NULL, 0`.

  **Return Value**

  If the function succeeds, the return value is `TRUE`. If the function fails, the return value is `FALSE`. To get extended error information, call `GetLastError`.

- `IOCTL_LINK_ENABLE_HEADER_MODE`

  Switches the device into header mode.

  **Parameters**

  - `lpInBuffer, nInBufferSize`
    InBuffer is not used. Specify `NULL, 0`.

  - `lpOutBuffer, nOutBufferSize`
    OutBuffer is not used. Specify `NULL, 0`.

  **Return Value**

  If the function succeeds, the return value is `TRUE`. If the function fails, the return value is `FALSE`. To get extended error information, call `GetLastError`.

- `IOCTL_LINK_DISABLE_HEADER_MODE`

  Switches the device back to stream mode.

  **Parameters**

  - `lpInBuffer, nInBufferSize`
    InBuffer is not used. Specify `NULL, 0`.

  - `lpOutBuffer, nOutBufferSize`
    OutBuffer is not used. Specify `NULL, 0`.

  **Return Value**

  If the function succeeds, the return value is `TRUE`. If the function fails, the return value is `FALSE`. To get extended error information, call `GetLastError`.

- `IOCTL_LINK_GET_INFO`

  Retrieves the driver revision.

  **Parameters**

  - `lpInBuffer, nInBufferSize`
    InBuffer is not used. Specify `NULL, 0`.

  - `lpOutBuffer, nOutBufferSize`
    OutBuffer points to an arrayof characters. This array will be filled with a \0-terminated C-string containing information about the revision of the driver. OutBuffer could look like this:

    ```
    driver V1.2\0
    ```

    Other transputer link interface drivers and future versions of the B004 driver return additional information.

  **Return Value**

  If the function succeeds, the return value is `TRUE`. The function fails if the caller supplied insufficient space to hold the string. The return value will then be `FALSE`. To get extended error information, call `GetLastError`.

## 7.6 CloseHandle

The CloseHandle function closes an open object handle.

```
BOOL CloseHandle (HANDLE hObject);
```

**Parameters**

- `hObject`
  Identifies an open object handle.

**Return Value**

If the function succeeds, the return value is `TRUE`. If the function fails, the return value is `FALSE`. To get extended error information, call `GetLastError`.

**Remarks**

`CloseHandle` invalidates the specified object handle, decrements the object's handle count, and performs object retention checks. Once the last handle to an object is closed, the object is removed from the operating system. Use `CloseHandle` to close handles returned by calls to the `CreateFile` function. Closing an invalid handle raises an exception. This includes closing a handle twice, not checking the return value and closing an invalid handle, and using `CloseHandle` on a handle returned by `FindFirstFile`.

## 7.7 Writing Applications For Multiple Link Interfaces

The generic device I/O interface supplied by Windows NT can be a big advantage when writing programs dealing with link interfaces. Under DOS, it was necessary to have several versions of each program, each dealing with one specific link interface hardware. Under NT, the same program is able to support any link interface hardware, as long as its driver presents the same software interface to the application. To achieve this, all the drivers must support the same set of ioctl codes.
Creating this common software interface is the reason why the file `link.h` does not only contain control codes valid for the B004 driver, but also ioctl codes used by other link interfaces. A driver may of course decide not to support some of the functions (and instead return an error if an application tries to use it), but it should not use the same ioctl codes for other purposes or with other parameter handling. An example for this would be the ioctl codes for changing link speed. The B004 driver cannot change the speed used by its hardware because this is selected by jumpers, so it simply returns an error indicating that this function is not available.
Any transputer link driver written by IMP will support this convention. If you are planning to write a new link interface driver and need some additional ioctl codes not yet defined in `link.h`, feel free to contact IMP (see below) and have your ideas included.

# 8 Header Mode

In addition to the usual stream mode the driver supports a special header mode. When operating in this mode, the driver automatically takes care of the `ISERVER` protocol. Because a transputer cannot be booted using this protocol, the driver supports two ioctl functions for enabling and disabling header mode.
When header mode is active, each data block handed down to `WriteFile` is interpreted as the data belonging to one iserver block. The driver prepends a 2 byte header specifying the length of the data to the block and writes both header and data to the link.
When receiving data from the link, the driver always interprets the first two bytes arriving as the length of the data to follow. It then reads as many bytes from the link as stated by the header and writes them to the buffer supplied by the caller of `ReadFile`. The caller must still supply a length parameter to the `ReadFile` call, but this parameter is only checked to prevent the driver from writing beyond the end of the users buffer. `ReadFile` returns the number of data bytes read.
Using header mode results in less calls to the driver during read operations and thus increases performance.

# 9 Troubleshooting

Common errors that might occur are:

- The driver does not start
  The registry might not be set up properly or the driver might not be able to allocate resources (either port addresses or an interrupt).

- The driver cannot be opened by applications
  Ensure the driver is started and you supplied the correct filename to CreateFile.

- The link can be opened, but data transfers always time out
  Make sure the link speeds used by the board and the transputer on the other end match. Does the transputer accept any data, i.e. has it been reset?