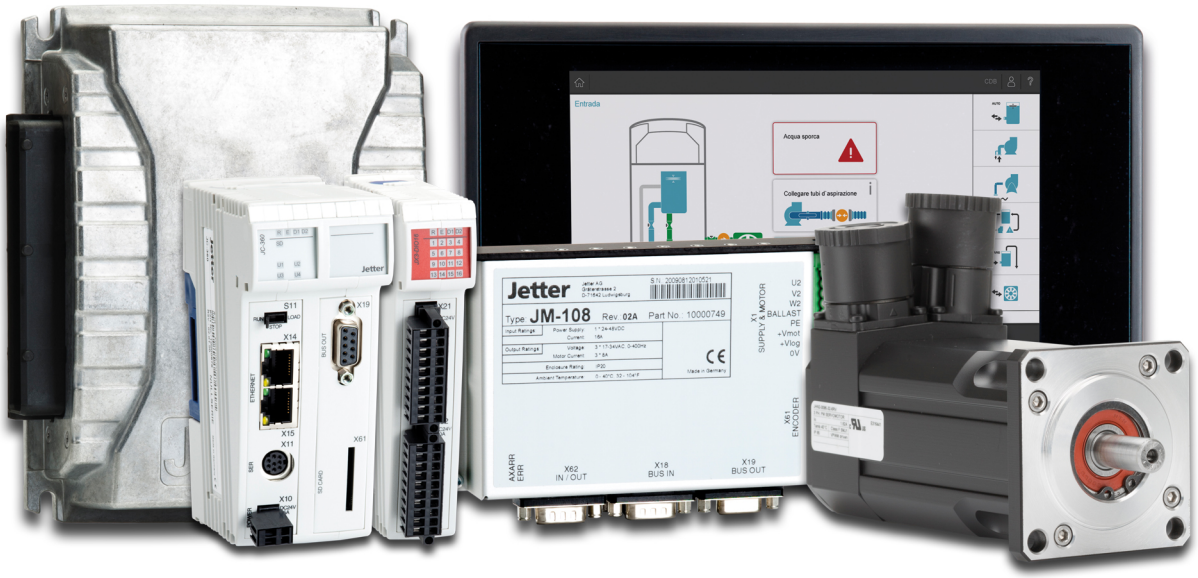
Jetter automation



Application-Oriented Manual

File System

60881632

We automate your success.

Item # 60881632
Revision 1.10
April 2017 / Printed in Germany

Table of Contents

1 File system

Introduction

This chapter covers the file system. The file system lets you access the files located on the internal flash disk drive, SD memory card or USB flash drive.

When problems occur, a good understanding of the file system is very helpful.

Note

Exercise extreme caution when dealing with the file system, especially with system files. Defective system files may cause your device to refuse to boot.

Some files may be protected against read/write access or deletion. This is normal behavior. Some of these files are virtual files, such as firmware images, or protected files, such as EDS files.

File categories

The files of the file system are categorized as follows:

- System directories or system files used by the operating system
- Files accessible to the user

Table of contents

1.1 Directories

System directories

The system directories cannot be deleted. System directories even survive formatting.

Controllers

Separate directory names by a slash "/", not by a backslash "\".

Directory	Description
/System	■ System configuration ■ System information
/SD	■ Root directory of the SD memory card
/USBx	■ Root directory of the USB flash drive x

HMIs

Separate directory names by a backslash "\", not by a slash "/".

Directory	Description
\System	■ System configuration ■ System information ■ Splash screen (boot image) ■ Screenshot
\SD	■ Root directory of the SD memory card
\USB	■ Root directory of the USB flash drive
\App	■ Directory for applications
\Data	■ Directory for data
\Windows	■ Windows CE system directory
\	■ RAM disk drive

Table of contents

Controller directories

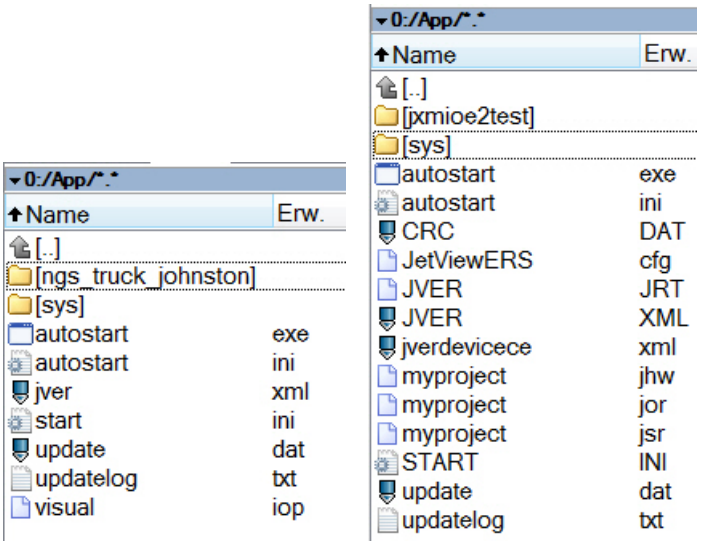
Directory */SD*	***/SD***
	When you insert an SD memory card into the SD card slot on the HMI, the dynamic directory named */SD* is created. When no data storage medium is inserted, this directory is not visible.
Directory */System*	***/System***
	This directory holds system-relevant files, such as the kernel, co-processor firmware, configuration data, EDS, etc.
Directory */USBx*	***/USBx***
	When you insert a USB flash drive into USB port x, a dynamic directory named */USBx* is created. When no data storage medium is inserted, this directory is not visible.

Directories referring to HMIs

Directory *\App*

\App

This directory holds application and visualization data. The screenshot below shows two different ways to store the STX application: Either in the file **ngs_truck_johnston** (left) or in the file **jxmioe2test** (right).



The screenshot to the left shows the file system of the old S platform. The visualization application is stored to an *.iop file. In the given example it is the file **visual.iop**.

The screenshot to the right shows the file system of the new CE platform. This platform does not use *.iop files. JetViewSoft creates several visualization files instead.

Note!

Copy all application and visualization files to the folder **App** and not to the folder **Data**. Failure to do so will slow down the boot process, see directory *\Data*.

start.ini

This text file defines which application will be started.

\App\sys

This directory holds the interpreter of the STX programming language and of the visualization software. **Do not make any changes here!**

autostart.exe

This application lets you update the operating system. **Do not make any changes here!**

Further, this file lets you start the device plus its visualization feature.

updatelog.txt

This is a log file which is created during an OS update.

Directory *Data*

Data

This directory holds the HMI's bulk data. The HMI lets you store parameter or configuration files to this directory.

Important Note!

Larger amounts of data can be stored to this data partition. To speed up system launch, this partition will be mounted a short instance, if needed, after launching the STX application. Therefore, the STX application must not be stored to this partition.

Directory *SD*

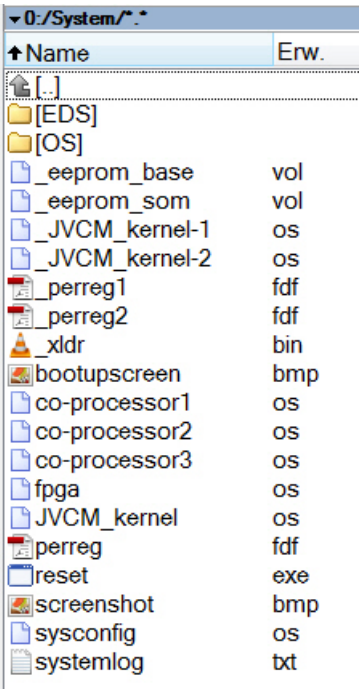
SD

When you insert an SD memory card into the SD card slot on the HMI, the dynamic directory named *\SD* is created. When no data storage medium is inserted, this directory is not visible.

Directory *System*

System

This directory holds system-relevant files, such as the kernel, co-processor firmware, configuration data, EDS, etc.



bootupscreen.bmp

This file is a 16-bit bmp file (r5, g6, b5) which is displayed while the device is booting.

You may create an image of your own and replace this file.

co-processor1

This virtual file holds the firmware of a hidden co-processor controlling most of the interactions with the user (buttons, buzzer, background lighting, etc.).

reset.exe

Deleting this file triggers the HMI to reboot immediately. You can use this function in batch files, for example, which, after complete processing, require automatic rebooting.

Directory *USB*

USB

When you insert a USB flash drive into the HMI, a dynamic directory named *\USB* is created. When no data storage medium is inserted, this directory is not visible.

Directory *Windows*

Windows

This directory holds the Windows CE files. **Do not make any changes here!**

1.2 Properties

Introduction

This chapter covers the properties of the file system. The file system distinguishes between internal flash disk drive, SD memory card, and USB flash drives.

General properties

The following properties apply to the internal flash disk drive, SD memory card, and USB flash drive:

- 8 files max. to be opened simultaneously
- Only apply lower case for directory and file names.
- When the device creates a file, it assigns its date and time.
- Date, time, and file size are not available for all system files.

Table of contents

Flash disk - Properties

Capacity	The memory size is dependent on the respective device.
Controllers: **Properties**	The internal flash disk drive has got the following properties: ▪ Up to 7 directory levels and 1 file level are allowed. ▪ Differentiation between upper and lower case. ▪ Directory and file names with a length of up to 63 characters are possible. ▪ All characters except "/" and ".." are permitted for directory and file names ▪ User/access administration for a maximum number of 31 locks and 33 users.
HMIs: **Properties**	The internal flash disk drive has got the following further properties: ▪ Up to 7 directory levels and 1 file level are allowed. ▪ Upper- and lower-case are not distinguished. ▪ Directory and file names are permitted to have a total length of 63 characters. ▪ All characters except "\" and ".." are permitted for directory and file names ▪ The location of the folders **App** and **Data** is on the flash disk drive. ▪ There is no user/access administration.

SD memory card - Properties

Capacity	The memory size is dependent on the respective device.
Properties	The SD memory card has got the following properties: ▪ The SD memory card must be compatible with FAT 16. ▪ Directory and file names of 260 characters' length max. can be used. ▪ The following characters are not permitted in directory and file names: "/", "\", ":", "*", "?", "<", ">" and "|" ▪ There is no user/access administration. Jetter AG cannot guarantee the proper functioning of all SD memory cards available on the market.

USB flash drive - Properties

Capacity	The memory size is dependent on the respective device.
Properties	The USB flash drive has got the following properties: ■ The USB flash drive must be compatible with FAT 16 or FAT 32. ■ Directory and file names of 260 characters' length max. can be used. ■ The following characters are not allowed in directory and file names: "/", "\", ":", "*", "?", "<", ">" and "|" ■ There is no user/access administration. Jetter AG will only guarantee for USB flash drives which they have made available as an option.

1.3 User administration

Introduction

The file system for the internal flash disk lets you define authorization for access (locks) to directories, and set up users.

For each user, you can set individual access rights (keys).

Users are not allowed to access directories and files for which they do not have the required key. In case of an FTP/IP connection, these directories and files are not displayed.

Prerequisites

Administrator rights are required for user administration.

Properties

The properties of user administration are as follows:

Property	Max. value
Number of users	33
Number of predefined users	2
Length of a user name	31 alphanumeric characters
Password length	31 alphanumeric characters
Number of keys for read access	31
Number of keys for write access	31
Number of predefined keys	2

Files

You can make settings for user administration in three files located in the directory **System**:

File	Function
flashdisklock.ini	Assignment of locks to directories
keys.ini	Assignment of names to locks/keys
users.ini	Administration of users

These files are always existing. They cannot be deleted, but only modified or overwritten.

Restrictions

Please take the following restrictions into account:

- User administration can only be applied to the internal flash disk. It cannot be applied to SD cards and USB flash.
- If user administration has been assigned to a file, its contents are readable at once. The settings become active only after a reboot.

Contents

Administration of users

Introduction

The configuration file **/System/users.ini** lets you manage the user administration for the file system.

Prerequisites

If you want to use names for the keys, you must make them known to the device beforehand. Therefore, set up the names first as described in *Setting up names for keys/locks* (see page 22).

Administration of users

To manage user administration, proceed as follows:

Step	Action
1	Establish an FTP connection to the device. Log on as administrator.
2	Open the file **/System/users.ini**.
3	Enter the required information.
4	Save the changed file to the device.
5	Reboot the device.

Result: The changed user administration settings are now enabled.

Structure of the configuration file

This configuration file is a text file the entries of which are grouped into several sections.

- For each user a separate section is to be created.
- In these sections values can be set which are then used by the file system.
- You can insert blank lines as required.
- The following characters precede a comment line: "!", "#" or ";".

Sections

The sections are named *[USER1]* through *[USER33]*. Here, the user name and the related password, as well as read and write permissions are specified.

Example:

```
[USER4]
NAME=TestUser3
PW=testpass
READKEYS=5,openLock2,10,11
WRITEKEYS=openLock2,10,11
SYSKEYS=
```

NAME	
In the given example	TestUser3
Description	User's login name
Allowed values	A maximum of 31 alphanumeric characters
In case of illegal value or missing entry	User administration settings are not made
PW	
In the given example	testpass
Description	User's login password
Allowed values	A maximum of 31 alphanumeric characters
In case of missing entry	The user is allowed to log in without password
READKEYS	
In the given example	5,openLock2,10,11
Description	Keys for read access (read keys)
Allowed values	1 ... 31 (or corresponding names)
In case of missing entry	No read keys are assigned to the user
WRITEKEYS	
In the given example	openLock2,10,11
Description	Keys for write access (write keys)
Allowed values	1 ... 31 (or corresponding names)
In case of missing entry	No write keys are assigned to the user
SYSKEYS	
Description	No function assigned; reserved for future extensions

Factory settings/predefined users and keys

Introduction

Two predefined users with set rights have been predefined in the file system. It is not possible to delete these two users. In the user administration only the password can be changed for these two users.

Factory settings

The factory settings include the content of the configuration file in the controller as follows:

```
[USER1]
NAME=admin
PW=admin
READKEYS=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
WRITEKEYS=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
SYSKEYS=

[USER33]
NAME=system
PW=system
READKEYS=2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
WRITEKEYS=2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
SYSKEYS=
```

Predefined users

All keys are available to the user *admin* who has, therefore, both read and write access to all directories and files.

All keys except for key **1** are available to user *system* as well.

Predefined keys

Two out of the 31 keys have a predefined function.

Lock/key	Function
1	■ Ethernet configuration ■ User administration
2	■ Operating system update of the CPU ■ Operating system update of JX2 and JX3 modules

Assigning locks

Introduction

The configuration file **/System/flashdisklock.ini** is used to assign locks to directories located on the flash disk. Only users with the corresponding key are allowed to read or write (delete) files and subdirectories located in these directories.

Prerequisites

If you want to use names for the locks, you must make them known to the device beforehand. Therefore, set up the names first: *Setting up names for keys/locks* (see page 22).

Assigning locks

To assign a lock to a directory, proceed as follows:

Step	Action
1	Establish an FTP connection to the device; when doing so, log in with administrator rights.
2	Open the file **/System/flashdisklock.ini**.
3	Adjust the file entries.
4	Save the adjusted file back to the device.
5	Reboot the device.

Result: A lock is assigned to this directory.

Structure of the configuration file

This configuration file is a text file containing one section.

- In this section values can be set which are then used by the file system.
- Specify each directory with its lock number in an individual line.
- You can insert blank lines as required.
- The following characters precede a comment line: "!", "#" or ";".

Section

The section is named *[LOCKS]*. Here, locks are assigned to directories in accordance with the following rule:

Directory=Lock

Example:

```
[LOCKS]
test1=0
test1/sub1=2
test1/sub2=5
test2=userlock2
```

Lock numbers

The lock numbers have got the following properties:

- The valid lock numbers are 0 ... 31.
- Lock number 0: No lock is assigned to this directory. You can access this directory without any special permissions.
- You can use numbers or previously defined names.

Assigning names to keys/locks

Introduction

Keys/locks are consecutively numbered from 1 through 31. To provide ease of handling, a name can be assigned to each key/lock combination. These names are assigned in the configuration file **/System/keys.ini**.

Assigning the names

To assign names to keys/locks, proceed as follows:

Step	Action
1	Establish an FTP connection to the device; when doing so, log in with administrator rights.
2	Open the file **/System/keys.ini**.
3	Enter the required information.
4	Save the adjusted file back to the device.
5	Reboot the device.

Result:

The names are available now. The names can now be used when assigning locks and managing user accounts.

Structure of the configuration file

This configuration file is a text file containing one section.

- In this section values can be set which are then used by the file system.
- Each key is specified with its name in an individual line.
- You can insert blank lines as required.
- The following characters precede a comment line: "!", "#" or ";".

Section

The section is named *[KEYS]*. Here, names are assigned to keys/locks in accordance with the following rule:

KEYxx=Name

xx: Number of the key (01 ... 31)

Example:

```
[KEYS]
KEY01=Admin
KEY02=System
KEY03=
KEY04=
KEY05=service
...
KEY31=
```

Names for locks and keys	For names the following definitions are true: ▪ A maximum of 15 alphanumeric characters ▪ Lock and key must have the same name.

1.4 Reviewing the flash disk capacity used

Introduction

You can view the application scope of the internal flash disk.
Details on the allocation of the application scope are given in this chapter.

Contents

Flash disk capacity used

README

You can view the application data area of the internal flash disk.

You can see the capacity used of the application data area from the file **/System/flashdiskinfo.txt**.

Example

In this example, the fictive capacity used of a flash disk in a JetControl 340 (4 MB) is shown:

```
Name  : flash disk
Date  : 25.11.2008
Time  : 15:04
Tracks: 64

Track   0:  sectors: 128  (used:  81 / blocked:  47 / free:   0)
Track   1:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   2:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   3:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   4:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   5:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   6:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   7:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   8:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track   9:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  10:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  11:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  12:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  13:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  14:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  15:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  16:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  17:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  18:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  19:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  20:  sectors: 128  (used:  64 / blocked:  64 / free:   0)
Track  21:  sectors: 128  (used:  85 / blocked:  43 / free:   0)
Track  22:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  23:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  24:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  25:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  26:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  27:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  28:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  29:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  30:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  31:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  32:  sectors: 128  (used: 128 / blocked:   0 / free:   0)
Track  33:  sectors: 128  (used: 105 / blocked:   0 / free:  23)
Track  34:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
```

```
Track  35:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  36:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  37:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  38:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  39:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  40:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  41:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  42:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  43:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  44:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  45:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  46:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  47:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  48:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  49:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  50:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  51:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  52:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  53:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  54:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  55:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  56:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  57:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  58:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  59:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  60:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  61:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  62:  sectors: 128  (used:   0 / blocked:   0 / free: 128)
Track  63:  sectors: 128  (used:   0 / blocked:   0 / free: 128)

Total:  sectors: 8192  (used: 4175 / blocked: 154 / free: 3863)

Used   :  2120900 byte
Blocked:    78232 byte
Free   :  1962404 byte
Total  :  4161536 byte
```

Elements of info file

Tracks and sectors represent the administration units of the flash disk. The info file comprises the following elements:

Element	Description
`Name`	Dedicated name of the flash disk
`Date/Time`	Point in time when the flash disk was formatted last
`Tracks`	Total number of tracks
`Track xx: sectors: 128`	Assignment of sectors of a track
`Total: sectors:`	Overall statistical data of the sectors
`Used`	Total number of used bytes
`Blocked`	Total number of blocked bytes
`Free`	Total number of available bytes
`Total`	Total size of the flash disk

States of the sectors

The smallest administrative unit of the flash disk, i.e. the sector, may assume the following states:

State	Description
`Used`	The sector is occupied by data.
`Blocked`	The sector is no longer occupied, but can not yet be used due to administrative reasons.
`Free`	The sector is not occupied and can be used.

1.5 Formatting and checking

Introduction

This chapter covers the following topics:

- Formatting the flash disk
- Formatting the SD card
- Checking the SD card
- Formatting the USB flash drive
- Checking the USB flash drive

The internal flash disk needs not be checked using a separate function, since it provides maximum safety of its administrative structures by design.

Functioning principle

When the device boots up, its OS system checks the content of the control register. The control register is part of the file system.

Depending on the value contained in this register the following functions are carried out:

- Formatting the flash disk
- Formatting the SD card
- Formatting the USB flash drive
- Checking the SD card
- Checking the USB flash drive

Register number

The number of the control register is 202936.

Table of contents

Formatting the flash disk

Introduction

In the following cases, reformatting the flash disk is required:

- When you upload an OS version that has got another flash disk format
- When information for flash disk administration has been destroyed

Consequences

- All files and directories located in the user area will be deleted!
- Formatting will not affect system files and directories.

Formatting the flash disk

To have the device format the internal flash disk, proceed as follows:

Step	Action
1	Switch the device ON.
2	Enter value -999720373 (0xc4697a4b) into the control register 202936 of the file system.
3	Switch the device OFF.
4	Switch the device ON.

Result: During the boot process the flash disk is formatted and the control register 202936 is set to **0**.

Formatting the SD card

Introduction

In the following cases, reformatting the SD card is required:

- When information for SD card administration has been destroyed

Consequences

All files and directories on the SD card will be deleted!

Formatting the SD card

To have the device format the SD card, proceed as follows:

Step	Action
1	Switch the device ON.
2	Enter value -748362163 (0xd364e64d) into the control register 202936 of the file system.
3	Switch the device OFF.
4	Switch the device ON.

Result: During the boot process the SD card is formatted and the control register 202936 is set to **0**.

Formatting the USB flash drive

Introduction

Sometimes it might be necessary to reformat the USB flash drive. This might be the case when information for USB flash drive administration has been destroyed.

Effect

All files and directories on the USB flash drive will be deleted!

Formatting

To format the USB flash drive proceed as follows:

Step	Action
1	Switch the device ON.
2	Enter value (0x8f3d5185) into the control register of the file system.
3	Switch the device OFF.
4	Switch the device ON.

Result: During the boot process of the device the USB flash drive is formatted and the control register is set to 0.

Checking the SD card

Introduction

In the following cases, checking the SD card for faults is required:

- When the device was switched off during access to the SD card

Consequences

- All files and directories on the SD card will be checked and errors, if any, will be fixed.
  Following such a check, the administrative structures on the SD card will be in consistent condition.
- Depending on the SD card size and the number of files and directories the boot process duration may extend to several minutes.

Checking the SD card

To have the device check the SD card, proceed as follows:

Step	Action
1	Switch the device ON.
2	Enter value 748371092 (0x2c9b3c94) into the control register 202936 of the file system.
3	Switch the device OFF.
4	Switch the device ON.

Result: While booting, the device checks the SD card. The value in the control register remains unchanged so that the card is checked whenever the Device is rebooted.

Restrictions

This function **repairs** the administrative structures on the SD card in order that it can be used further. However, it may happen that the device cannot restore in all cases data of a file, which, for example, has been written incompletely.

Checking the USB flash drive

Introduction

Sometimes it might be necessary to check the USB flash drive for errors. This might be the case if the device was de-energized while accessing the USB flash drive.

Effect

- All files and directories on the USB flash drive will be checked and errors, if any, will be fixed.
  Following such a check, the administrative structures on the USB flash drive are in consistent condition.
- Depending on the USB flash drive capacity and the number of files and directories to be checked the boot process may take several minutes.

Check

To have the device check the USB flash drive for errors proceed as follows:

Step	Action
1	Switch the device ON.
2	Enter value (0x17dbd42a) into the control register of the file system.
3	Switch the device OFF.
4	Switch the device ON.

Result: During the boot process of the device, the USB flash drive is being checked. The value in the control register remains unchanged so that the stick is checked whenever the device is rebooted.

Restrictions

This function only **repairs** the administrative structures on the USB flash drive so that it can be used further. However, it may happen that data of a file, which has been written incompletely, can't be restored.

2 FTP server

Introduction

The FTP server allows access to directories and files using an FTP client. The files can be stored to the following storage media:

- Internal Flash disk
- SD memory card
- USB flash drive

This chapter covers the login process and describes the commands supported by the FTP server.

FTP clients

The user has the option of using a command line FTP client, which comes with many PC operating systems, or graphic FTP tools.

Amount of possible connections

The FTP server is able to manage up to four FTP connections simultaneously. Any additional FTP client, which tries to connect with the FTP server, will get no response to its request for establishing a connection.

Supported commands

The FTP server supports standardized commands. For more information refer to:

- FTP server help menu; connect with FTP server and enter the command *help* or *help binary*.
- In the Web, search for FTP and commands

If you do not wish to care about commands, we recommend using an FTP program, such as TotalCommander.

R 202930

Web status

The Web status register displays all available functions in bit-coded mode.

Meaning of the individual bits	
Bit 0	**FTP server** 1 = available
Bit 1	**HTTP server** 1 = available
Bit 2	**E-mail** 1 = available
Bit 3	**Data file function** 1 = available
Bit 4	**Modbus/TCP** 1 = existing
Bit 5	**Modbus/TCP** 1 = available
Bit 7	**FTP client** 1 = available

Module register properties	
Type of access	Read
Value after reset	Depending on options purchased

Required programmer's skills

To be able to use the functions described in this chapter, the programmer must be familiar with the following subjects:

- File system
- IP networks
- FTP commands

Contents

Logon

Logon	To be able to access the file system via FTP, the FTP client must log on when the connection is established. ▪ As **Server Name** enter the IP address of the device. ▪ As **User Name** enter your user name, e.g. admin. ▪ As **Password** enter your password, e.g. admin.
Factory settings	The factory settings of the <Produktname einfügen> include one user account: NAME=admin PW=admin

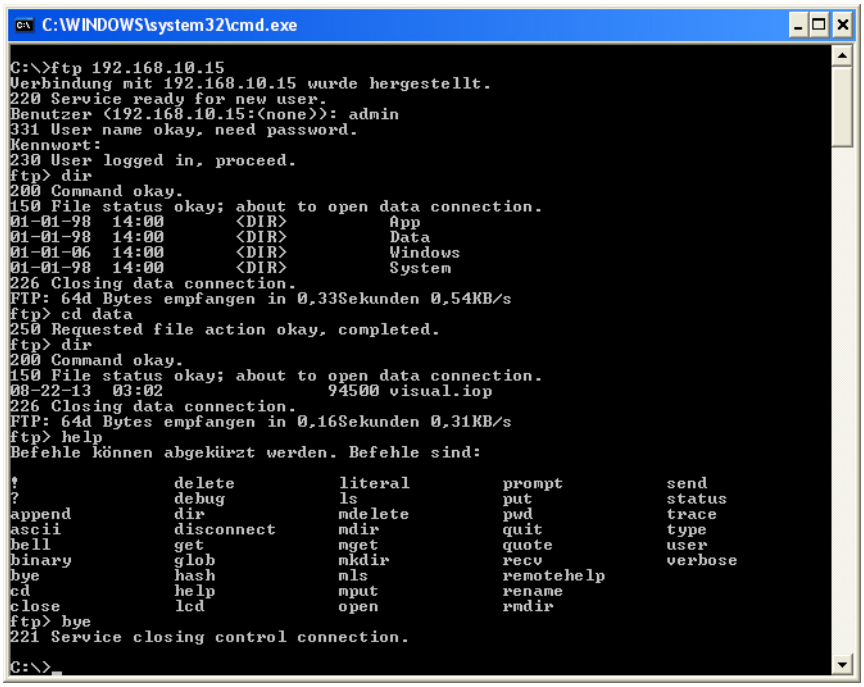
Example: Windows FTP client

Task

Carry out the following tasks using an FTP client, for example, the one which comes with Windows XP:

- Launch the FTP client by opening a connection and entering the IP address.
- Log on as user *admin* with password *admin*.
- Use *dir* to display the content of the current directory.
- Enter *cd app* to change to directory *app*.
- Use *dir* to display the content of the current directory.
- Enter *help* to view all available commands.
- Terminate the session and the FTP client using the command *bye*.

Action



3 HTTP server

Introduction

A standard browser is sufficient for accessing the HTTP server.

The browser is for reading and displaying files which have been downloaded to the controller via FTP.

Here, it may be necessary to enter the user name and password to have access to certain pages (depending on the file system configuration).

Default file names

The default file names are **index.htm** and **index.html**.

Supported file types

The following file types are supported:

- *.htm, *.html, *.shtml
- *.txt, *.ini
- *.gif, *.tif, *.tiff, *.bmp, *.wbmp
- *.jpg, *.jpe, *.jpeg, *.png
- *.xml
- *.js, *.jar, *.java, *.class, *.cab
- *.ocx
- *.pdf, *.zip, *.doc, *.rtf
- *.css
- *.wml, *.wmlc, *.wmls, *.wmlsc
- *.ico, *.svg

Enabling the HTTP server feature

On the controller <Produktname einfügen>, the feature HTTP Server is always enabled.

That is, bit 1 in Web Status register 202930 is always set.

Required programmer's skills

To be able to use the functions described in this chapter, the user must be familiar with the following:

- File system
- IP networks

Contents

3.1 Server Side Includes

Introduction

The HTTP server features *Server Side Includes* (SSI). This function is for showing present real-time controller values on an HTML page.

Rules

You must specify a **Name Space** tag at the beginning of the HTML page that is to contain the real-time controller values.

This **Name Space** is for defining the namespace used in the HTML page.

In the body section of the HTML page the **Data** tags are specified.

Updating real-time controller values

When the HTML page is uploaded to the browser, the HTTP server once replaces the **Data** tags by actual real-time controller values.

To refresh the controller values, the HTML page must be reloaded over and over again.

The user triggers reloading by entering the controller address and the name of the required page, e.g. *http://192.168.10.209/Homepage/SSI/ssiTimeAndDate.htm*.

Contents

First entry in the HTML file

Configuration

The **Name Space** must be the first entry in the HTML file. It has got the following structure:

```
<NS:DTAG xmlns:NS=http://jetter.de/ssi/jetcontrol/
```

with **NS** representing the namespace. The namespace is a character string with a maximum length of 63 characters.

The namespace introduced here will be re-used for the subsequent **Data** tags. The remaining parts of the line are preassigned and have to be specified in exactly the same way.

In the following examples, the namespace applied is **JC**.

Inserting real-time controller values

Introduction

Actual real-time controller values are integrated into parameter entries within the sections via tag functions. This way, the contents respectively states of registers, text registers, inputs, outputs and flags can be displayed.

Tag delimiters

All tags start and end with defined strings. Between these tag delimiters, the variables are defined.

Delimiter	String
Tag start	<JC:DTAG
Tag end	/>

Variable definition

The variable definition in a tag contains attributes which are used to set, for example, how the value of a variable is to be displayed:

name	
Description	Variable name
Comments	Code letter followed by the variable number
Example	`name="R1000023"`

type	
Description	Variable type of notation
Example	`type="REAL"`

format	
Description	Representation format
Comments	Refer to format definition
Example	`format="+0####.###"`

factor	
Description	Factor by which the real-time controller value is multiplied
Comments	Multiplication is executed prior to adding the offset.
Example	`factor="1.5"`

offset	
Description	Value which is added to the real-time controller value
Comments	Multiplication by the factor is executed prior to adding the value to the real-time controller value.
Example	`offset="1000"`

Format definition

You can define the representation of variables by means of their attribute.

- The number of digits/characters used for representing a variable can be defined by the character "#".
- Prefix "0" sets the output of leading zeros. This applies to the register types INT, INTX and REAL.
- Prefix "+" sets the output of a sign. This applies to the register types INT and REAL.
- Prefixing a blank sets the output of a blank. This applies to the register types INT and REAL.

Registers/text registers

The variable name begins with a capital "R" followed by the register number.
The following types are possible:

Type	Notation
INT	Integer, decimal
INTX	Integer, hexadecimal
INTB	Integer, binary
BOOL	Register content = 0 --> Display: 0 Register content != 0 --> Display: 1
REAL	Floating point, decimal
STRING	Text register

Standard type: INT

Example:

```
JC:DTAG name="R1000250" type="REAL" format="+0####.###"
factor="3.25" offset="500" /
```

Result:

This instruction causes the contents of register 1000250 to be multiplied by 3.25 and then added to product 500. The result appears in the Web browser with sign and at least five integer positions before the decimal point. If need be, five leading zeros are added. Furthermore, three decimal positions are added.

Flags

The variable name begins with a capital "F" followed by the flag number.
The following types are possible:

Type	Notation
BOOL	Flag = 0 --> Display: 0 Flag = 1 --> Display: 1
STRING	Flag = 0 --> Display: FALSE Flag = 1 --> Display: TRUE

Standard type: BOOL

Example:

```
<JC:DTAG name="F100" type="STRING" format="#" />
```

Result:
The state of flag 100 is displayed as string "T" or "F".

Inputs

The variable name begins with a capital "I" followed by the input number.
The following types are possible:

Type	Notation
BOOL	Input = 0 --> Display: 0 Input = 1 --> Display: 1
STRING	Input = 0 --> Display: OFF Input = 1 --> Display: ON

Standard type: BOOL

Example:

```
<JC:DTAG name="I201200308" type="STRING" />
```

Result:
The state of input 201200308 on the CPU is displayed as string "ON" or "OFF".

Outputs

The variable name begins with a capital "O" followed by the output number.
The following types are possible:

Type	Notation
BOOL	Output = 0 --> Display: 0 Output = 1 --> Display: 1
STRING	Output = 0 --> Display: OFF Output = 1 --> Display: ON

Standard type: BOOL

Example:

```
<JC:DTAG name="O201100308" />
```

Result:
The state of output 201100308 is inserted as "1" or "0".

Access via pointer register

Access via pointer register is realized by inserting the capital letter "P" in front of the variable name. In each case the value of the variable is displayed whose number corresponds to the content of the register specified in the

variable name.

Examples:

```
<JC:DTAG name="PR1000300" />
```

Result: The content of the register is displayed whose number is contained in register 1000300.

```
<JC:DTAG name="PF1000300" />
```

Result: The state of the flag is displayed whose number is contained in register 1000300.

```
<JC:DTAG name="PI1000300" />
```

Result: The state of the input is displayed whose number is contained in register 1000300.

```
<JC:DTAG name="PO1000300" />
```

Result: The state of the output is displayed whose number is contained in register 1000300.

Access via pointer register and offset

To specify the number of the variable to be displayed, it is also possible to add a constant value or another register content to the pointer register value

Examples:

```
<JC:DTAG name="PR1000300 + 100" />
```

Result: The content of the register is displayed whose number results from the addition of the content of register 1000300 and value 100.

```
<JC:DTAG name="PR1000300 + R1000100" />
```

Result: The content of the register is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JC:DTAG name="PF1000300 + 100" />
```

Result: The state of the flag is displayed whose number results from the addition of the content of register 1000300 and value 100.

```
<JC:DTAG name="PF1000300 + R1000100" />
```

Result: The state of the flag is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JC:DTAG name="PI1000300 + 100" />
```

Result: The state of the input is displayed whose number results from the addition of the content of register 1000300 and the value 100.

```
<JC:DTAG name="PI1000300 + R1000100" />
```

Result: The state of the input is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

```
<JC:DTAG name="PO1000300 + 100" />
```

Result: The state of the output is displayed whose number results from the addition of the content of register 1000300 and the value 100.

```
<JC:DTAG name="PO1000300 + R1000100" />
```

Result: The state of the output is displayed whose number results from the addition of the content of register 1000300 and the content of register 1000100.

Example of an HTML page

Task

Insert current real time controller values into an HTML page.

It should then be possible to display the HTML page in a browser using the *Server Side Include* feature.

Action

```
<JC:DTAG xmlns:JC="http://jetter.de/ssi/jetcontrol" />
<html>

<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgID" content="FrontPage.Editor.Document">
<title>Index</title>
</head>

<body>
Hello World, 
<p>Actual controller values can be inserted into an html page like
this: </p>
<p>Register 201000 = <JC:DTAG name="R201000" type = INT
format="+####" />,
or Hex: 0x<JC:DTAG name="R201000" type="INTX" format="0###" />,
or rather this way: <JC:DTAG name="R201000" type="BOOL" />, if only
Boolean is queried.
But binary is also possible: <JC:DTAG name="R201000" type="INTB"
format=######## />b. </p>
<p>Strings could also be defined "<JC:DTAG name="R201000"
type="STRING" />". </p>
<p>A real number looks as follows: <JC:DTAG name="R1001500"
type="REAL" />
or this way: <JC:DTAG name="R1001500" type="REAL" factor="1.3"
format="###.##" />. </p>
<p>The value of a flag is represented as follows: <JC:DTAG name="F10"
/>
or <JC:DTAG name="F10" type="STRING" />. </p>
<p>With inputs and outputs, it is done the same way: <JC:DTAG
name="I100000205" type="BOOL" />
or <JC:DTAG name="I100000205" type="STRING" />. </p>
<p>R201000 = <JC:DTAG name="R201000" type="INT"
format="+0##########" /> </p>
<p>Regards </p>
<p>Yours JetControl</p>
</body>

</html>
```

Storage location

Now store the HTML page to the file system of the controller.

4 FTP client

The FTP client

The FTP client allows access from within the application program to files and directories of a remote network device. To this end, the FTP client communicates with the FTP server of this network device.

Functions

The following functions are possible:

- Creating directories in the remote file system.
- Deleting directories in the remote file system.
- Copying files from the local file system into the remote file system.
- Copying files from the remote file system into the local file system.

Requirements

- To be able to use the FTP client feature basic knowledge of FTP connections and file systems is required.
- The IP address of the FTP server must be known.
- If the IP address of the FTP server is not known, name resolution through a DNS server must be possible.
- User name and password for logging on at the FTP server must be known.
- For programming this feature JetSym version 4.3 or higher is required.

Processing within the application program

- The controller completes only one FTP access at a time.
- The corresponding task in the application program stops at the command until the access is completed.
- During this time other tasks in the application program are processed.
- While an FTP access of a task is being processed, all other tasks which invoke an FTP command are blocked until the FTP access is completed.

R 202930

Web status

The Web status register displays all available functions in bit-coded mode.

Meaning of the individual bits	
Bit 0	**FTP server**
	1 = available
Bit 1	**HTTP server**
	1 = available
Bit 2	**E-mail**
	1 = available
Bit 3	**Data file function**
	1 = available
Bit 4	**Modbus/TCP**
	1 = existing
Bit 5	**Modbus/TCP**
	1 = available
Bit 7	**FTP client**
	1 = available

Module register properties	
Type of access	Read
Value after reset	Depending on options purchased

Contents

4.1 Programming

Introduction

The FTP client allows to access files and directories on a network device from within the application program. For this purpose, function calls are used. These function calls are included in the programming language of the controller. To program this feature, proceed as follows:

Step	Action
1	Initialize the FTP client
2	Open the connections to the FTP servers
3	Transfer data
4	Terminate the connections

Restrictions

While the controller is processing one of the functions of the FTP client, tasks supporting the FTP client should not be stopped through `TaskBreak` or restarted through `TaskRestart`. Otherwise the controller fails to complete this function which will block new function calls by the FTP client.

Contents

Initializing the FTP client

Introduction

At each application program start, the FTP client must be initialized at least once.

Function declaration

```
Function FtpInitialize():Int;
```

Return value

The following return value is possible:

Return value	
0	Always

How to apply this function

The function is used and its return value assigned to a variable for further utilization in the following way:

```
Result := FtpInitialize();
```

Operating principle

The controller processes this function in the following steps:

Step	Description
1	The controller closes all open connections of the FTP client.
2	The controller initializes all OS-internal data structures of the FTP client.

Establishing a connection to the FTP server

Introduction

Before data can be sent or received, a connection to the FTP server must be established first. When establishing the connection, the client logs in to the FTP server by a user name and a password (login).

Function declaration

```
Function FtpConnect(Const Ref ServerAddr: String,
                    Const Ref UserName: String,
                    Const Ref PassWord: String):Handle;
```

Function parameters

Description of the function parameters:

Parameter	Value	Remarks
ServerAddr	IP address or name	Name resolution by DNS server
UserName	User name for logon	Login
Password	Password for logon	Login

Return value

If the return value was positive, the connection could be established and login was successful. If the return value was **0**, an error occurred and the connection could not be established.

Return value	
> 0	A positive return value must be stored in a variable. It must be made available as a handle at activating the following functions: ■ `Terminating a connection` ■ `Reading a file from the FTP server` ■ `Writing a file to the FTP server` ■ `Deleting a file from the FTP server` ■ `Changing a directory on the FTP server` ■ `Creating a directory on the FTP server` ■ `Deleting a directory from the FTP server` ■ `Determining the current directory on the FTP server`
0	Error when establishing a connection or logging in to the FTP server

Operating principle

The task stops at the program line until the connection is established or the timeout set for the FTP client has elapsed.

This function is processed in the following steps:

Step	Description	
1	The controller tries to establish a TCP/IP connection to the FTP server.	
2	**If ...**	**... then ...**
	... the network client has accepted the connection, ...	... proceed with step 3.
	... the connection could not be established and the timeout has not elapsed yet, ...	... proceed with step 1.
	... an error has occurred or the timeout has elapsed, ...	... the function is terminated and value **0** is returned.
3	The controller logs on to the FTP server with its user name *Administrator* and password *AdminPassword*.	
4	**If ...**	**... then ...**
	... the FTP server has accepted the connection, ...	... the function is terminated and a positive value is returned as handle for further access to this connection.
	... the FTP server has not accepted the connection, for example because of an invalid user name or wrong password, ...	... the function is terminated and value **0** is returned.

Related topics

- **Terminating a connection** (see page 55)

Terminating a connection

Introduction

Clear all connections which are no longer required as this will reduce PLC load for managing connections.

Function declaration

```
Function FtpDisconnect(FtpConnection:Handle):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Remarks
FtpConnection	Handle	Value returned by the function `FtpConnect()`

Return value

The following return values are possible:

Return value	
0	Connection terminated and deleted
-1	Invalid handle
-2, -3	Communication error, there is, for example, no response from FTP server

Related topics

- **Establishing a connection to the FTP server** (see page 53)

Reading a file

Introduction

This function lets you read the content of a file from a remote network node and copy it to the local file system of the controller.

Function declaration

```
Function FtpFileRead(FtpConnection:Handle,
                     Const Ref ServerFile: String,
                     Const Ref ClientFile: String):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Remarks
FtpConnection	Handle	Value returned by the function `FtpConnect()`
ServerFile	File name	Name of the file in the file system of the FTP server, which the controller is to read
ClientFile	File name	File name, as which the controller is to save the file read in the local file system

Return value

If the returned value is negative, an error has occurred. If the returned value is **0**, the controller was able to read the file and store it locally.

Return value	
0	No error
-1	Invalid handle
-2, -6	Error when storing the file locally
-3, -5, -7, -8	Communication error, there is, for example, no response from FTP server
-4	Error message from FTP server, for example, file does not exist

Operating principle

In the following cases, the task is not processed further after issuing the function call:

- The controller must first read the file, e. g. **ServerTestFile.txt** and save it to the local file system as, e. g., **LocalTestFile.txt**.
- An error has occurred.

This function is processed in the following steps:

Step	Description	
1	The controller sends a command to the FTP server that the content of the file **ServerTestFile.txt** must be transmitted.	
2	The controller receives the contents of the file **ServerTestFile.txt**.	
3	The controller writes the contents to the file **LocalTestFile.txt**.	
4	**If ...**	**... then ...**
	... no errors have occurred, ...	... the file has been copied successfully, the function is terminated and value **0** is returned.
	... errors have occurred, ...	the function is terminated and a negative value is returned.

File names

- The function parameter for the local file may contain the path to this file, e.g. `"/Data/TestFiles/LocalTestFile.txt"`.
- If the file system of the remote network node supports this, the function parameter for the file located on the FTP server can also contain the path to this file. Otherwise, the directory must be set beforehand using the command `FtpDirChange()`.
- The file system supports both options.

Related topics

- **Writing to a file** (see page 58)

Writing to a file

Introduction

This function lets you copy the content of a file belonging a local file system to a file belonging to the file system of a remote network node.

Function declaration

```
Function FtpFileWrite(FtpConnection:Handle,
                      Const Ref ServerFile: String,
                      Const Ref ClientFile: String):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Remarks
FtpConnection	Handle	Value returned by the function `FtpConnect()`
ServerFile	File name	File name as which the FTP server is to save the written file
ClientFile	File name	Name of the file in the local file system, the content of which the controller is to send to the FTP server

Return value

If the returned value is negative, an error has occurred. If the returned value is **0**, the controller was able to read the file and store it to the file system of the remote network node.

Return value	
0	No error
-1	Invalid handle
-2	Error when reading the local file, e.g. file does not exist
-3, -5, -8	Communication error, there is, for example, no response from FTP server
-4, -7	Error message from the FTP server, e.g. file cannot be created

Operating principle

In the following cases, the task is not processed further after issuing the function call:

- The controller must first read the file, e. g. **LocalTestFile.txt** and save it to the file system of the remote network node as, e. g., **ServerTestFile.txt**.
- An error has occurred.

This function is processed in the following steps:

Step	Description	
1	The controller sends a command to the FTP server that the content of the file **ServerTestFile.txt** must be saved.	
2	The controller sends the contents of the file **LocalTestFile.txt**.	
3	The FTP server writes the contents to the file **ServerTestFile.txt**.	
4	**If ...**	**... then ...**
	... no errors have occurred, ...	... the file has been copied successfully, the function is terminated and value **0** is returned.
	... errors have occurred, ...	the function is terminated and a negative value is returned.

File names

- The function parameter for the local file may contain the path to this file, e.g. `"/Data/TestFiles/LocalTestFile.txt"`.
- If the file system of the remote network node supports this, the function parameter for the file located on the FTP server can also contain the path to this file. Otherwise, the directory must be set beforehand using the command `FtpDirChange()`.
- The file system supports both options.

Related topics

- **Reading a file** (see page 56)

Deleting a file

Introduction

This function lets you remove a file from the file system of a remote network node.

Function declaration

```
Function FtpFileRemove(FtpConnection:Handle,
                       Const Ref ServerFile: String):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Remarks
FtpConnection	Handle	Value returned by the function `FtpConnect()`
ServerFile	File name	Name of the file to be removed.

Return value

If the returned value is negative, an error has occurred. If the returned value is **0**, the file could not be removed from the file system of the remote network node.

Return value	
0	No error
-1	Invalid handle
-2	Communication error, there is, for example, no response from FTP server
-3	Error message from FTP server, for example, file does not exist

Operating principle

In the following cases, the task is not processed further after issuing the function call:

- The FTP server must first delete the file **ServerTestFile.txt**. Please note: This file name serves as an example only.
- An error has occurred.

This function is processed in the following steps:

Step	Description	
1	The controller sends a command to the FTP server that the file **ServerTestFile.txt** must be deleted.	
2	The FTP server deletes the file **ServerTestFile.txt**.	
3	**If ...**	**... then ...**
	... no errors have occurred, ...	... the file has been deleted successfully, the function is terminated and value **0** is returned.
	... errors have occurred, ...	the function is terminated and a negative value is returned.

File names

- The function parameter for the local file may contain the path to this file, e.g. `"/Data/TestFiles/LocalTestFile.txt"`.
- If the file system of the remote network node supports this, the function parameter for the file located on the FTP server can also contain the path to this file. Otherwise, the directory must be set beforehand using the command `FtpDirChange()`.
- The file system supports both options.

Changing directories

Introduction

This function lets you remove the current directory from the file system of a remote network node.

Function declaration

```
Function FtpDirChange(FtpConnection:Handle,
                      Const Ref ServerDir: String):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Remarks
FtpConnection	Handle	Value returned by the function `FtpConnect()`
ServerDir	Directory name	Name of the directory into which the user wants to change

Return value

If the returned value is negative, an error has occurred. If the returned value is **0**, the system has managed to change directories.

Return value	
0	No error
-1	Invalid handle
-2	Communication error, there is, for example, no response from FTP server
-3	Error message from the FTP server, e.g. directory does not exist

Operating principle

In the following cases, the task is not processed further after issuing the function call:

- The FTP server must first change directories.
- An error has occurred.

This function is processed in the following steps:

Step	Description	
1	The controller sends a command to the FTP server that it has to change to a subdirectory.	
2	The FTP server changes directories.	
3	**If ...**	**... then ...**
	... no errors have occurred, ...	... the new directory is set, the function is terminated and value **0** has been returned.
	... errors have occurred, ...	the function is terminated and a negative value is returned.

Directory names

- If the file system of the remote network node supports this, the function parameter for the directory located on the FTP server can also contain the complete path including several subdirectories leading to this directory.
- If this feature is not supported, the user must navigate from one directory level to the next until the desired directory is reached. This is done using the command `FtpDirChange()`.
- The file system of the device supports both options.

Related topics

- **Determining the current directory** (see page 68)

Creating a directory

Introduction

This function lets you create a new directory from the file system of a remote network node.

Function declaration

```
Function FtpDirCreate(FtpConnection:Handle,
                      Const Ref ServerDir: String):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Remarks
FtpConnection	Handle	Value returned by the function `FtpConnect()`
ServerDir	Directory name	Name of the directory to be created

Return value

If the returned value is negative, an error has occurred. If the returned value is **0**, the directory could successfully be created in the file system of the remote network node.

Return value	
0	No error
-1	Invalid handle
-2	Communication error, there is, for example, no response from FTP server
-3	Error message from FTP server, e.g. directory already exists

Operating principle

In the following cases, the task is not processed further after issuing the function call:

- The FTP server must first create a subdirectory.
- An error has occurred.

This function is processed in the following steps:

Step	Description	
1	The controller sends a command to the FTP server that it has to create a subdirectory.	
2	The FTP server creates the directory.	
3	**If ...**	**... then ...**
	... no errors have occurred, ...	... the new directory has been created, the function is terminated and value **0** is returned.
	... errors have occurred, ...	... the function is terminated and a negative value is returned.

Directory names

- If the file system of the remote network node supports this, the function parameter for the directory located on the FTP server can also contain the complete path including several subdirectories leading to this directory.
- If this feature is not supported, the user must navigate from one directory level to the next until the desired directory is reached. This is done using the command `FtpDirChange()`.
- The file system of the device supports both options.

Restrictions regarding the file system of a JetControl

If you specify a directory with the corresponding path as function parameter, all directories up to the directory you want to create must exist. Recursive creation of several directories is not supported.

Example:

```
Result := FtpDirCreate(FtpHandle,
                       '/DataFiles/TextFiles/Release');
```

To be able to create the folder **Release** in the directory tree */DataFiles/TextFiles* the directories must already exist.

Related topics

- **Deleting directories** (see page 66)

Deleting directories

Introduction

This function lets you remove a directory from the file system of a remote network node.

Function declaration

```
Function FtpDirRemove(FtpConnection:Handle,
                      Const Ref ServerDir: String):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Remarks
FtpConnection	Handle	Value returned by the function `FtpConnect()`
ServerDir	Directory name	Name of the directory to be deleted

Return value

If the returned value is negative, an error has occurred. If the returned value is **0**, the directory could successfully be removed from the file system of the remote network node.

Return value	
0	No error
-1	Invalid handle
-2	Communication error, there is, for example, no response from FTP server
-3	Error message from the FTP server, e.g. directory does not exist

Operating principle

In the following cases, the task is not processed further after issuing the function call:

- The FTP server must first remove the subdirectory.
- An error has occurred.

This function is processed in the following steps:

Step	Description	
1	The controller sends a command to the FTP server that it has to remove the subdirectory.	
2	The FTP server removes the subdirectory.	
3	**If ...**	**... then ...**
	... no errors have occurred, ...	... the directory is removed, the function is terminated and value **0** is returned.
	... errors have occurred, ...	the function is terminated and a negative value is returned.

Directory names

- If the file system of the remote network node supports this, the function parameter for the directory located on the FTP server can also contain the complete path including several subdirectories leading to this directory.
- If this feature is not supported, the user must navigate from one directory level to the next until the desired directory is reached. This is done using the command `FtpDirChange()`.
- The file system of the device supports both options.

Related topics

- **Creating a directory** (see page 64)

Determining the current directory

Introduction

This function lets you determine the current directory in the file system of a remote network node.

Function declaration

```
Function FtpDirPrint(FtpConnection:Handle,
                     Ref str: String):Int;
```

Function parameters

Description of the function parameters:

Parameter	Value	Remarks
FtpConnection	Handle	Value returned by the function `FtpConnect()`
str	String address	Current directory with path specification

Return value

If the returned value is negative, an error has occurred. If the returned value is **0**, the current directory could successfully be determined in the file system of the remote network node.

Return value	
0	No error
-1	Invalid handle
-3	Communication error, there is, for example, no response from FTP server
-4	Error message sent by the FTP server
-5	Invalid response from server

Operating principle

In the following cases, the task is not processed further after issuing the function call:

- The FTP server must first determine the actual directory.
- An error has occurred.

This function is processed in the following steps:

Step	Description	
1	The controller sends a command to the FTP server that it has to determine the current directory.	
2	The FTP server transmits the actual directory with path specification.	
3	**If ...**	**... then ...**
	... no errors have occurred, ...	... the variable contains the complete path of the current directory, the function is terminated and value **0** is returned.
	... errors have occurred, ...	the function is terminated and a negative value is returned.

Related topics

- **Changing directories** (see page 62)

4.2 Registers

Introduction

This chapter describes the registers on the controller which contain status information of the FTP client. These registers can be used for debugging or diagnostic purposes. However, they can't be used for other functions such as establishing or terminating a connection.

Contents

Register numbers

Introduction

Data of one connection each are displayed within the registers of a coherent register block. Two other registers show the status of the command being executed by the application program. The basic register number of these registers is dependent on the controller.

Register numbers

Basic register number	Register numbers
320000	320000 ... 320101

Determining the register number

In this chapter, only the last three figures of a register number are specified. e.g. MR 002. To determine the complete register number, add to this module register number the basic register number of the corresponding device, for example 320000.

Registers - Overview

FTP client module registers - Overview

Register	Description
MR 000	Number of open connections
MR 002	Timeout in seconds
MR 003	Port number of the FTP server
MR 004	Index in the connection table
MR 005	Connection handle
MR 006	IP address of the FTP server
MR 007	Port number of the FTP server
MR 008	IP address of FTP client
MR 009	Port number of FTP client
MR 100	Processing status on part of FTP client
MR 101	Task ID

Registers - Description

Introduction

Established connections are managed by the operating system of the controller in a list. Module registers MR 004 or 005 are used to copy connection data into registers MR 006 through MR 009.

MR 000

Number of open connections

The value in this register shows how many connections are currently open.

Module register properties		
Reading values	0 ... 2,147,483,647	Number of connections

MR 002

Timeout

To this register, write the timeout of the FTP client at accessing the FTP server.

Module register properties		
Values	0 ... 2,147,483,647	in seconds
Value after reset	20	

MR 003

Port number of the FTP server

The value in this register shows the IP port number of the FTP server.

Module register properties	
Values	0 ... 65,535
Value after reset	21

MR 004

Index in the connection table

The index of the connection table is entered into this register. If a connection has been established for a given index, the connection handle can be seen in module register MR 005 and connection data in module registers MR 006 through MR 009.

Module register properties	
Values	0 ... [MR 000] - 1
Value after reset	-1

MR 005

Connection handle

This register is for entering the connection handle. If a connection has been established for a given index, the connection handle can be seen in module register MR 004 and connection data in module registers MR 006 through MR 009.

Module register properties	
Values	0 ... 2,147,483,647

MR 006

IP address of the FTP server

The value in this register shows the IP address of the FTP server.

Module register properties	
Access	Read
Takes effect	if MR 004 >= 0

MR 007

Port number of the FTP server

The value in this register shows the port number of the FTP server.

Module register properties	
Access	Read
Takes effect	if MR 004 >= 0

MR 008

IP address of FTP client

The value in this register shows the IP address of the FTP client.

Module register properties	
Access	Read
Takes effect	if MR 004 >= 0

MR 009

Port number of FTP client

The value in this register shows the port number of the FTP client.

Module register properties	
Access	Read
Takes effect	if MR 004 >= 0

MR 100

Processing status on part of FTP client

This register lets you track the processing status on part of FTP client.

Module register properties		
Values	0	No access at the moment
	1	Parameters are being handed over to the FTP client of the controller
	2	The FTP client communicates with the FTP server.
	3	Access completed
Access	Read	

MR 101

Task ID

This register shows the ID of the task which is processing an FTP client function at that moment.

Module register properties		
Values	0 ... 99	Task ID
	255	None of the tasks is carrying out an FTP function.
Value after reset	255	
Access	Read	

5 AutoCopy - Automatic copying of controller data

Introduction

This chapter describes the AutoCopy function which allows to copy data within the controller and/or between the controller and an FTP server, the connected expansion modules and a controller within the network. To this end, create a command file which is then stored along with the data either to the SD card, or to the USB flash drive. This command file is automatically processed by the controller during the boot process.

Functions within the local file system

AutoCopy executes the following functions:

- Storing registers and flags to a file
- Restoring registers and flags from a file
- Creating directories
- Deleting directories
- Copying files
- Deleting files

Functions within the file system of an FTP server

AutoCopy executes the following functions:

- Copying files from the FTP server
- Copying files to the FTP server
- Deleting files
- Changing directories
- Creating a directory
- Deleting directories

Areas of application

Basically, AutoCopy is used in the following scenarios:

- Where remote maintenance is not possible
- Where there is no PC on site
- If the operator is not able or should not be allowed to make modifications to the plant

The AutoCopy function lets you:

- Modify the application program
- Modify the application data
- Modify the controller configuration
- Operating system update (controller, modules on the system bus, network devices)
- Duplicate a control system

Prerequisites

For automatic copying of controller data, the following prerequisites must be fulfilled:

- The programmer must be familiar with the file system.
- The programmer must have basic knowledge in the area of FTP application.

config.ini - Example

This is an example of a configuration file **config.ini** with an entry *AutoCopyIni*.

```
;Copyright (c) 2009 by Jetter AG, Ludwigsburg, Germany

[IP]
Address     = 192.168.  1.  1
SubnetMask = 255.255.255.  0
DefGateway  =   0.  0.  0.  0
DNSServer   =   0.  0.  0.  0

[HOSTNAME]
SuffixType = 0
Name        = JetControl350

[PORTS]
JetIPBase   = 50000
JVMDebug    = 52000

[FILES]
AutoCopyIni = /SD/project_name/autocopy.ini
```

AutoCopyIni - Note

- The AutoCopy function only makes sense, if the data to be copied have been stored to the SD card or to the USB flash drive. This means that the root directory is */SD/* or */USB/*.
- The file **autocopy.ini** can be stored to any directory.
- Instead of **autocopy.ini**, you can name the file arbitrarily.

Designation

In this description, *Full Name* means the name of the file or directory including its full path.

Table of contents

5.1 Operating principle

Introduction

This chapter describes how to start and execute the AutoCopy function.

Contents

Launching the AutoCopy feature

Introduction

The AutoCopy function can only be executed when the controller is booting (i.e. after startup).

Prerequisites

You have created the command file and stored it to the respective directory.

If the entry *AutoCopyIni* is not available in the configuration file **config.ini** the name of the command file and of the directory is set as follows:

	Value	Remarks
File name	autocopy.ini	All lower case letters
Directory	/SD/	Root directory on the SD card
Directory	/USB/	Root directory on the USB flash drive

- The file **autocopy.ini** can be stored to any directory.
- Instead of **autocopy.ini**, you can name the file arbitrarily.

In this case, it is prerequisite that the configuration file **config.ini** contains the entry *config.ini*. This entry defines the directory and file name of the command file.

Launching the AutoCopy feature

To launch the AutoCopy function, proceed as follows:

Step	Action	
1	Disconnect the controller from the power supply.	
2	**If ...**	**... then ...**
	... you use an SD card,	... insert the SD card completely into the SD slot.
	... you use a USB flash drive,	... insert the USB flash drive into the USB port of the controller.
3	Set the mode selector to *LOAD* position.	
4	Switch the controller on.	
5	Wait for the red LED **D1** to be lit and for the green LED **R** and the yellow LED **SD**  to flash slowly by approximately 1 Hz.	
⇨	**Result:** The controller executes the AutoCopy function.	
6	Wait for the red LED **D1** and for the green LED **R** to flash slowly by approximately 1 Hz.	
⇨	**Result:** The AutoCopy process is completed.	

Executing AutoCopy commands

Introduction

During the boot process in AutoCopy mode the controller executes the commands contained in the command file.

Restrictions

In AutoCopy mode the following restrictions of controller functions apply:

- The controller does not execute the application program.
- Communication with the controller is not possible.
- When the AutoCopy function is completed the controller must be restarted.

Executing AutoCopy commands

The OS of the controller processes the AutoCopy function in the following steps:

Step	Description
1	The controller opens the command file that is specified by the entry *AutoCopyIni* in the configuration file **/System/config.ini**.
2	The controller reads the values from section `[OPTIONS]`.
3	The controller reads the command and its parameters from section `[COMMAND_1]`, processes it and writes the results, if any, into the log file.
4 ... n	The controller processes the other commands in ascending order up to the number given in section `[OPTIONS]`.
n+1	The controller calculates the statistic values for all command results and writes them into the log file.

LEDs in AutoCopy mode

During the boot process of the controller, the OS status LEDs indicate the following:

Step	Description					
1						
	R	**E**	**D1**	**D2**	**SD/**	**State**
	4Hz	4Hz	4Hz	4Hz	OFF	Reset
2						
	R	**E**	**D1**	**D2**	**SD/**	**State**
	1Hz	OFF	OFF	ON	OFF	The boot loader is running and is checking the OS. For controllers not having got a boot loader: The controller initializes the OS.

3

R	E	D1	D2	SD/	State
1Hz	OFF	OFF	OFF	OFF	The OS reads the settings of the DIP switch on the backplane module and checks whether an Ethernet switch exists.

4

R	E	D1	D2	SD/	State
1Hz	ON	OFF	OFF	OFF	The OS initializes the realtime clock, the Ethernet port and the file system.

5

R	E	D1	D2	SD/	State
1Hz	ON	ON	OFF	/ OFF	The OS initializes the modules on the JX3 and JX2 system bus and the SD card.

6

R	E	D1	D2	SD/	State
1Hz	OFF	ON	OFF		The command file of the AutoCopy function is being processed.

7a

R	E	D1	D2	SD/	State
1Hz	OFF	1Hz	OFF	OFF	AutoCopy function is completed; no errors occurred.

7b

R	E	D1	D2	SD/	State
1Hz	ON	1Hz	OFF	OFF	AutoCopy function is completed; errors occurred.

Terminating AutoCopy function

Introduction

Only a restart of the controller terminates the AutoCopy function.

Prerequisites

Processing the AutoCopy command is completed.

Terminating AutoCopy function

To terminate the AutoCopy function, proceed as follows:

Step	Action
1	Disconnect the controller from the power supply.
2	The SD card or the USB flash drive can now be removed (not required).
3	Set the mode selector to *RUN* or *STOP* position.
4	Switch the controller on.

Result: The controller relaunches.

5.2 autocopy.ini - Structure

Introduction

This chapter covers the structure of the file **autocopy.ini** and the available commands.

File structure

This command file of the AutoCopy function is a text file the entries of which are grouped into several sections.

- In these sections you can set values then used by the AutoCopy function.
- You can insert blank lines as required.
- Introduce comment marks by "!", "#" oder ";".

Sections

The command file has two section types:

- In the *[OPTIONS]* section, you can make default settings. This file is unique.
- In the *[COMMAND_#]* section, you can specify the commands that are to be executed. The number of command section is limited to a value of 128.

Contents

Section [OPTIONS]

Introduction

In the `[OPTIONS]` section, you can make default settings. This section exists only once, preferably at the beginning of the file.

Example

```
[OPTIONS]
CommandCount = 14
LogFile      = /SD/autocopy.log
LogAppend    = 1
```

Elements of this section

The section consists of the following elements:

CommandCount	
In the given example	14
Function	Number of command sections that follow
Allowed values	> = 0
Illegal values	< 0
In case of illegal value or missing entry	0
LogFile	
In the given example	/SD/autocopy.log
Function	Complete name of the log file
Allowed values	■ All allowed file names ■ Directory exists
Illegal values	■ Incorrect filename ■ Non-existent directory
In case of illegal value or missing entry	The controller does not create a log file.
LogAppend	
In the given example	1
Function	Defines whether a new log file is to be created or whether it is to be appended to an existing one.
Allowed values	■ 0 = Delete file which may exist and create a new one ■ 1 = Append file to an existing one. If no file exists, the controller creates a new log file.
Illegal values	■ < 0 ■ > 1
In case of illegal value or missing entry	The controller re-creates the log file.

Command sections

Introduction

In these sections you can specify commands which are then executed by the AutoCopy function.

Example

```
[COMMAND_1]
Command        = DirCreate
Path           = /Homepage
ErrorAsWarning = 1

[COMMAND_2]
Command        = FileCopy
Source         = /SD/Index.htm
Destination    = /Homepage/index.htm

[COMMAND_3]
Command        = FtpConnect
ServerAddr     = 192.168.123.45
UserName       = admin
Password       = admin
```

Section names

The names of the sections consist of the `COMMAND_` string followed by a value. The value is between one and the value of the `CommandCount` entry from section `[OPTIONS]`.

Processing commands

The AutoCopy function processes the commands in order of their section names:

- Starting with the command under section `[COMMAND_1]`
- Ending with the command under the section with the value of entry `CommandCount` from section `[OPTIONS]`
- Each command section may only contain one command. Thus, you have to create an individual section for each command.

Troubleshooting

When an error occurs while a command is being processed, the controller makes a corresponding entry in the log file. For each command the user can set, whether the controller is to enter the error into the log file as `Error` or as `Warning`. Make this setting by the optional parameter `ErrorAsWarning`.

ErrorAsWarning	Entry into the log file
Parameter does not exist	Error
ErrorAsWarning = 0	Error
ErrorAsWarning = 1	Warning

File names

- The function parameter for the local file may contain the path to this file, e.g. `'Data/TestFiles/LocalTestFile.txt'`.
- If the file system supports this, the function parameter for the file located on the FTP server can also contain the path to this file. If this feature is not supported, the corresponding directory must be set beforehand using the command `FtpDirChange()`.
- The file system supports both options.

Available commands in the local file system

The following commands are available for access to the local file system:

Command = DirCreate	
Function	Creates a subdirectory
Parameter name	Path
Parameter value	Complete directory name
Allowed values	■ All valid directory names ■ Higher-level directories are available
Illegal values	■ Invalid directory name ■ Non-existent higher-level directory ■ Name of an already existing directory
In the event of an illegal value	The controller does not generate the directory. It enters the error into the log file.

Example	`[COMMAND_1]`
	`Command = DirCreate`
	`Path = /sub1`
	
	`[COMMAND_2]`
	`Command = DirCreate`
	`Path = /sub1/sub2`

Command = DirRemove	
Function	Removes a subdirectory
Parameter name	Path
Parameter value	Complete directory name
Allowed values	■ All valid directory names ■ The directory is empty
Illegal values	■ Invalid directory name ■ Directory is not empty
In the event of an illegal value	The controller does not delete the directory. It enters the error into the log file.

Example	`[COMMAND_8]`
	`Command = DirRemove`
	`Path = /sub1/sub2`

Command = FileCopy	
Function	This command is for copying a file.
Parameter name 1	Source
Parameter value 1	Complete name of the source file

Parameter name 2	Destination
Parameter value 2	Complete name of the destination file
Allowed values	■ All allowed file names ■ The destination directory does exist
Illegal values	■ Incorrect filename ■ Non-existent source file ■ Non-existent destination directory
In the event of an illegal value	The controller does not copy the file. It enters the error into the log file.

Example	`[COMMAND_1]`
	`Command = FileCopy`
	`Source = /SD/OS/JC-340_1.04.0.03.os`
	`Destination = /System/OS/op_system.os`
	
	`[COMMAND_2]`
	`Command = FileCopy`
	`Source = /SD/Manual.pdf`
	`Destination = /sub1/Manual.pdf`

Command = FileRemove	
Function	Deleting a file
Parameter name	Path
Parameter value	Complete name of the file
Allowed values	All allowed file names
Illegal values	Incorrect filename
In the event of an illegal value	The controller does not delete the file. It enters the error into the log file.

Example	`[COMMAND_5]`
	`Command = FileRemove`
	`Path = /sub1/Manual.pdf`

Command = DaFileRead	
Function	Transferring register values and flag states from a data file to the controller
Parameter name	DaFile
Parameter value	Complete name of the data file
Allowed values	All allowed file names for data files
Illegal values	■ Incorrect filename ■ Nonexistent data file
In the event of an illegal value	The data are not transmitted to the controller. The controller enters the error into the log file.

Example	`[COMMAND_12]`
	`Command = DaFileRead`
	`DaFile = /SD/Data/MyTestData.da`

Command = DaFileWrite

Function	This command is for storing register values and flag states to a data file.
Parameter name 1	DaFile
Parameter value 1	Complete name of the data file
Allowed values	■ All allowed file names for data files ■ The destination directory does exist
Illegal values	■ Incorrect filename ■ Non-existent destination directory
In the event of an illegal value	The controller does not generate the data file. It enters the error into the log file.
Parameter name 2	Append
Parameter value 2	Defines whether a new data file is to be created or it is to be appended to an existing one
Allowed values	■ 0 = Delete the data file which may exist and create a new data file ■ 1 = Append the file to an existing one. If no file exists, the controller creates a new data file
Illegal values	■ < 0 ■ > 1
In the event of an illegal value	A new data file will be created
Parameter name 3	Type
Parameter value 3	Defines whether registers or flags are to be stored
Allowed values	■ Registers ■ Flag
Illegal values	Values other than *Register* or *Flag*
In the event of an illegal value	The controller does not generate the data file. It enters the error into the log file.
Parameter name 4	First
Parameter value 4	Number of the first register or flag
Allowed values	All valid numbers from the memory area of the corresponding controller
Illegal values	Invalid numbers
In the event of an illegal value	The controller does not generate the data file. It enters the error into the log file.
Parameter name 5	Last

Parameter value 5	Number of the last register or flag
Allowed values	All valid numbers from the memory area of the corresponding controller which are equal to or greater than the value for *First*
Illegal values	■ Invalid numbers ■ Numbers less than *First*
In the event of an illegal value	The controller stores only one value (First).

Example	`[COMMAND_11]`
	`Command = DaFileWrite`
	`DaFile = /SD/MyTestData2.da`
	`Append = 0`
	`Type = Register`
	`First = 1000000`
	`Last = 1000000`
	
	`[COMMAND_12]`
	`Command = DaFileWrite`
	`DaFile = /SD/MyTestData2.da`
	`Append = 1`
	`Type = Flag`
	`First = 10`
	`Last = 20`
	
	`[COMMAND_13]`
	`Command = DaFileWrite`
	`DaFile = /SD/MyTestData2.da`
	`Append = 1`
	`Type = Register`
	`First = 1000001`
	`Last = 1000999`

Available commands for access via FTP

The following commands are available for access via network using FTP:

Command = FtpConnect	
Function	Establishing a connection to an FTP server
Parameter name 1	ServerAddr
Parameter value 1	IP address or name of FTP server
Allowed values	■ IP address of the FTP server ■ Name which can be resolved through DNS
Illegal values	■ IP address other than that of the FTP server ■ Name which cannot be resolved
Parameter name 2	UserName
Parameter value 2	User name for logging on at the FTP server
Parameter name 3	Password
Parameter value 3	Password for logging on at the FTP server
In the case of a illegal values	The controller does not establish the connection. It enters the error into the log file.

Example	`[COMMAND_1]`
	`Command = FtpConnect`
	`ServerAddr = 192.168.123.45`
	`UserName = admin`
	`Password = admin`

Restriction	Only one connection with an FTP server can be established at a time. The controller terminates the existing connection, before a connection to another FTP server is established.

Command = FtpFileRead	
Function	Copying file from FTP server into the local file system
Parameter name 1	ServerFile
Parameter value 1	Complete name of the source file in the FTP server
Parameter name 2	ClientFile
Parameter value 2	Complete name of the destination file in the local file system
Allowed values	■ All allowed file names ■ The destination directory does exist
Illegal values	■ Incorrect filename ■ Non-existent source file ■ Non-existent destination directory
In the event of an illegal value	The controller does not copy the file. It enters the error into the log file.

Example	`[COMMAND_8]`
	`Command = FtpFileRead`
	`ServerFile = /app/cantest/cantest.es3`
	`ClientFile = /SD/cantest3.es`

Command = FtpFileWrite	
Function	Copying the file from the local file system into the file system of the FTP server
Parameter name 1	ServerFile
Parameter value 1	Complete name of the destination file in the FTP server
Parameter name 2	ClientFile
Parameter value 2	Complete name of the source file in the local file system
Allowed values	■ All allowed file names ■ The destination directory does exist
Illegal values	■ Incorrect filename ■ Non-existent source file ■ Non-existent destination directory
In the event of an illegal value	The controller does not copy the file. It enters the error into the log file.

Example	`[COMMAND_5]`
	`Command = FtpFileWrite`
	`ServerFile = /System/OS/op_system.os`
	`ClientFile = /SD/OS/JC-340_1.09.0.00.os`

Command = FtpFileRemove	
Funktion	Deleting a file from the FTP server
Parameter name	ServerFile
Parameter value	Complete filename
Allowed values	All allowed file names
Illegal values	Incorrect filename
In the event of an illegal value	The controller does not delete the file. It enters the error into the log file.

Example	`[COMMAND_9]`
	`Command = FtpFileRemove`
	`ServerFile = /sub1/Manual.pdf`

Command = FtpDirChange	
Function	Changing the working directory in FTP server
Parameter name	ServerDir
Parameter value	Complete directory name
Allowed values	All valid directory names
Illegal values	Invalid directory name
In the event of an illegal value	The controller does not switch the directory. It enters the error into the log file.

Example	`[COMMAND_12]`
	`Command = FtpDirChange`
	`ServerDir = /Data/MyTestData`

Command = FtpDirCreate	
Function	Creating a subdirectory in the FTP server
Parameter name	ServerDir
Parameter value	Complete directory name
Allowed values	■ All valid directory names ■ Higher-level directories are available
Illegal values	■ Invalid directory name ■ Non-existent higher-level directory ■ Name of an already existing directory
In the event of an illegal value	The controller does not generate the directory. It enters the error into the log file.

Example	`[COMMAND_6]`
	`Command = FtpDirCreate`
	`ServerDir = /Data/MyTestData`

Restriction	If a directory with the corresponding path is specified as function parameter, all directories up to the directory to be created must exist. Recursive creation of several directories is not supported.

Command = FtpDirRemove	
Function	Clear the subdirectory in the FTP server
Parameter name	ServerDir
Parameter value	Complete directory name

Allowed values	■ All valid directory names ■ The directory is empty
Illegal values	■ Invalid directory name ■ Directory is not empty
In the event of an illegal value	The controller does not delete the directory. It enters the error into the log file.

Example	`[COMMAND_8]`
	`Command = FtpDirRemove`
	`ServerDir = /Data/MyTestData`

Example of a command file

Task

The JetControl 340 may serve as an example. Via various JX3 modules, it controls an already existing plant. In this plant, you want to enhance the functions.

To this end, the following modifications are required:

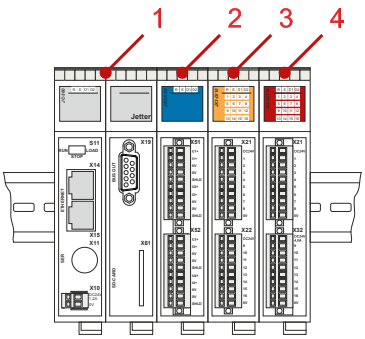
- Operating system update for the controller
- Operating system update for an analog output module
- New application program
- New values for some of the registers

Solution

You copy the required files to an SD card and create a command file for the AutoCopy function. Then you send this SD card along with a short instruction sheet to the plant operator. Once the update is completed, the operator is to return the SD card.

Sample configuration

This example is based on the following configuration:



Number	Part	Description
1	JC-340	Controller
2	JX3-AO4	Analog output module Module number: 02
3	JX3-DI16	Digital input module
4	JX3-DIO16	Digital output module

SD card contents

The following illustration shows the directory structure and the files on the SD card from the controller's point of view before the AutoCopy function is executed:

Following execution the log file **autocopy.log** has been added.

Command file

```
[OPTIONS]
CommandCount = 7
LogFile      = /SD/autocopy.log
LogAppend    = 0

# update operating system of controller
[COMMAND_1]
Command     = FileCopy
Source      = /SD/OS/JC-340_1.04.0.00.os
Destination = /System/OS/op_system.os

# update operating system of JX3-AO4 module
[COMMAND_2]
Command     = FileCopy
Source      = /SD/OS/JX3-AO4_1.01.0.00.os
Destination = /System/JX3-Module02/OS/system.os

# create user program directories
# probably already present - but to be sure ...
[COMMAND_3]
Command        = DirCreate
Path           = /app
ErrorAsWarning = 1

[COMMAND_4]
Command     = DirCreate
Path        = /app/userprogtest
```

```
# copy user program start file
[COMMAND_5]
Command     = FileCopy
Source      = /SD/UserProgs/start.ini
Destination = /app/start.ini

# copy user program
[COMMAND_6]
Command     = FileCopy
Source      = /SD/UserProgs/userprogtest.es3
Destination = /app/userprogtest/userprogtest.es3

# set registers and flags
[COMMAND_7]
Command     = DaFileRead
DaFile      = /SD/UserData/MyTestData.da
```

5.3 Log file

Introduction

This chapter covers the structure and contents of the log file into which the device enters the outcome of the respective commands.

Contents

File contents

Introduction

The log file is a plain text file. By making an entry into the command file, you define whether a log file is to be created or whether the device is to append the entries to an existing log file.

Example

```
JetControl AutoCopy log file 07.11.2008 09:14:09

 1: Ok      - FileCopy   /SD/OS/JC-340_1.04.0.00.os
                         /System/OS/op_system.os (345740 byte)
 2: Ok      - FileCopy   /SD/OS/JX3-AO4_1.01.0.00.os
                         /System/JX3-Module02/OS/system.os
                         (16832 byte)
 3: Warning - DirCreate  /app
 4: Ok      - DirCreate  /app/userprogtest
 5: Ok      - FileCopy   /SD/UserProgs/start.ini
                         /app/start.ini (63 byte)
 6: Ok      - FileCopy   /SD/UserProgs/userprogtest.es3
                         /app/userprogtest/userprogtest.es3
                         (169 byte)
 7: Error   - DaFileRead /SD/UserData/MyTestData.da

Command statistics:
  Total  : 7
  Ok     : 5
  Warning: 1
  Error  : 1
```

Description

When for each executed AutoCopy function a section is appended to an existing log file, the log file consists of three elements:

- The header contains date and time
- The following block contains information on the executed commands.
- Finally, it contains short statistics on command processing.

In the above example an error occurs when trying to create the directory */app* as this directory already exists. The device enters this error as a warning. When the device reads the DA file, an error also occurs. The device enters this error into the log file.

5.4 Data files

Introduction

This chapter covers data files where register and flag values are stored.

Contents

File format

Format

The data file consists of the following elements:

- Pure text file
- Each entry must be in a separate line of text
- Each line must be terminated by carriage return/line feed
- Comment lines must be preceded by ";"
- Each data file is to start with the entry *SD1001*.

Data lines

A data line consists of the following elements:

- ID of the variable at the beginning of the line
- Now follows the number of the variable separated by a blank or tab
- Then follows the value of the variable separated by a blank or tab

Variable ID	Variable type
FS	Flags
RS	Integer register
QA	Floating-point registers

Example

```
SD1001
; Data File - Jetter AG
;
; Registers 1000000 ... 1000005
RS    1000000    12345
RS    1000001    2
RS    1000002    -1062729008
RS    1000003    502
RS    1000004    50
RS    1000005    3
QS    1009000    3.14
;
; Flags 10 ... 13
FS    10    0
FS    11    1
FS    12    1
FS    13    0
```

6 Application program

Introduction

This chapter describes how to store the application program in the device. The user determines the program that is to be executed.

Required programmer's skills

This chapter requires knowledge on how to create application programs in JetSym and how to transmit them via the file system of the device.

Contents

Application program - Default path

Introduction

When uploading the application program from JetSym to the controller, this program is stored as a file to the internal flash disk. The device enters the path and file name into the file **start.ini** which is in the folder **app**.

Path and file name

In the folder **app**, JetSym, by default, creates a subdirectory and assigns the project name to it. Then, JetSym stores the application program to this subdirectory assigning the extension ***.esX** to it. The path and file names are always converted into lower case letters. **X** is a hardware-dependent placeholder. e.g. **es4** for JetControl 400 and **es9** for JetControl 900.

start.ini - Structure

This file is a text file with one section holding two entries:

Element	Description
[Startup]	Section name
Project	Path to the application program file relating to the folder **app**
Program	Name of the application program file

Example:

```
[Startup]
Project = test_program
Program = test_program.es9
```

The application program is loaded from the file **test_program.es9** which is located in the folder **app** in subdirectory *test_program*.

Related topics

- **Storing the application program** (see page 101)

Storing the application program to the SD memory card or the USB flash drive

Introduction

When uploading the application program from JetSym to the controller, the default storage for application programs is used.

If you want the device to read the application program from the SD card or from the USB flash drive, you have to configure the file path.

If you want to store the application program to another directory of the internal flash disk, proceed the same way.

Prerequisites

Only apply lower case for directory and file names.

Storing the application program to the SD card or the USB flash drive

If you want to store the application program to the SD card or USB flash drive, configure the device as follows:

Step	Action
1	Create an application program file by JetSym.
2	Create the desired directory on the SD card or the USB flash drive.
3	Store the application program file to the desired directory.
4	Write the path to the application program file and the program name into the file **start.ini** in the folder *app* on the internal flash disk of the device.

Result: On re-boot, the device loads the application program from the SD card or USB flash drive.

start.ini - Structure

This file is a text file with one section holding two entries:

Element	Description
[Startup]	Section name
Project	Path leading to the application program file
Program	Name of the application program file

Controller:

Example - SD card:

```
[Startup]
Project = /sd/testprogram
Program = test1.esx
```

Example - USB flash drive:

```
[Startup]
Project = /usb/testprogram
Program = test1.esx
```

HMI:

Example - SD card:

```
[Startup]
Project = \sd\testprogram
Program = test1.esx
```

Example - USB flash drive:

```
[Startup]
Project = \usb\testprogram
Program = test1.esx
```

The HMI JetView:

Example - SD card:

```
[Startup]
Project = \..\..\..\Storage Card
Program = test1.esx
```

Example - USB flash drive:

```
[Startup]
Project = \..\..\..\USBMemory
Program = test1.esx
```

Result:

The application program is loaded from the file **test1.esx** located in the subdirectory **testprogram** of folder *sd* on the SD card or in the folder *usb* on the USB flash drive.

Related topics

- **Application program - Default path** (see page 100)

Loading an application program

Introduction

At reboot of the application program via JetSym or after booting the device, the application program is loaded and executed via the file system.

Loading the application program by the OS of the controller

For this, mode selector S11 must be in *RUN* position.

The application program is loaded by the controller's OS as follows:

Step	Description
1	The OS reads the file **/app/start.ini** from the internal flash disk.
2	The OS evaluates the entry **Project**. It contains the path leading to the application program file.
3	The OS evaluates the entry **Program**. The entry contains the program name.
4	The OS loads the application program from the file **<Project>/<Program>**.

Loading the application program by the OS of the HMI

The application program is loaded by the OS of the HMI as follows:

Step	Description
1	The OS reads the file **\app\start.ini** from the internal flash disk.
2	The OS evaluates the entry **Project**. It contains the path leading to the application program file.
3	The OS evaluates the entry **Program**. It contains the program name.
4	The OS loads the application program from the file **<Project>/<Program>**.

Jetter automation

Jetter AG
Graeterstrasse 2
71642 Ludwigsburg | Germany

Phone +49 7141 2550-0
Fax +49 7141 2550-425
info@jetter.de
www.jetter.de

We automate your success.