FUJITSU COMPUTERS SIEMENS

Reliant UNIX ONLINE Documentation

IOCS V3.0

Administration, Configuration and Programming of Printers

Edition April 1993

Identification: U8347-J-Z815-1-7600

Copyright and Trademarks

1 Introduction

1.1 Layout of manual

Following the introductory chapter, this manual contains the following chapters:

- Administration of IOCS

  This chapter gives an overview of the process trace options and describes the commands used in the administration of IOCS.
- Configuration of IOCS

  This chapter describes the configuration of printers managed by IOCS.
- User interface for printers

  This chapter describes the functions for programming printers via IOCS.
- Compatible control sequences

  This chapter describes the set of control sequences that are sent to the printer in addition to the actual data to be printed and initiate a specific printer reaction (e.g. paper feed).
- Installation of IOCS

  This chapter describes the installation of IOCS.

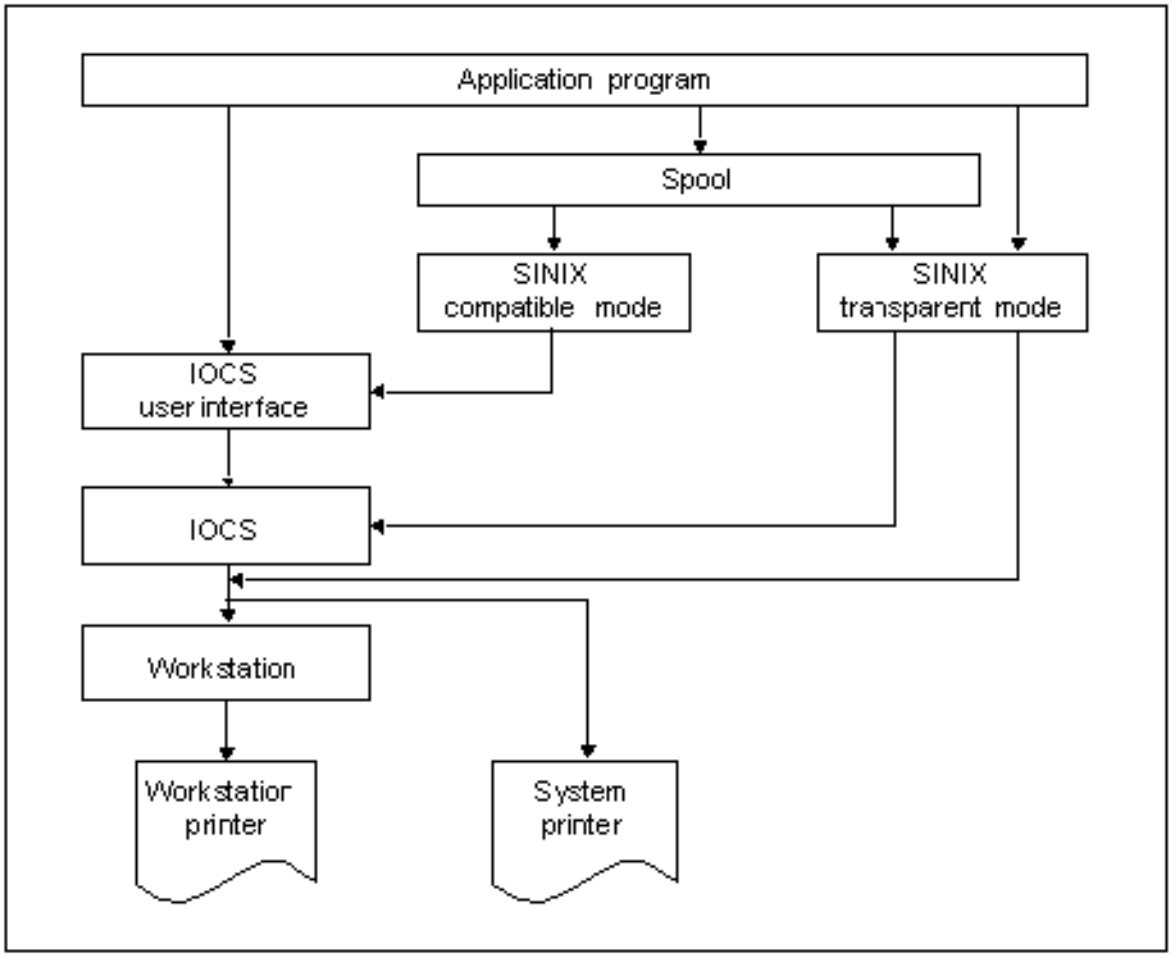
1.2 Target groups

Chapter "Administration of IOCS", Chapter "Configuration of IOCS" and Chapter "Installation of IOCS" are aimed at system administrators.

Chapter "User interface for printers" and Chapter "Compatible control sequences" are aimed at applications programmers experienced in programming printers.

1.3 Interfaces for connecting printers

The SINIX operating system provides various options for connecting printers. The options and system layers involved are described below. The following diagram provides a basic overview:



Application program

The application programmer's standard applications or individual print routines can be based on various programming interfaces, each which provides various services and each being supported differently in the system.

IOCS user interface

The IOCS user interface comprises special commands for printer programming (see Section "C programming interface"). These commands facilitate synchronization mechanisms, such as locking and asynchronous programming, in addition to compatible programming. Moreover, they provide an extended error handling compared to the SINIX® modes, so that the application can react to any errors that occur.

SINIX compatible mode

The familiar C commands are used in this mode. However, the programmer need not know the respective control sequences of the connected printer. More important are sequences from the set of compatible control sequences, which are converted by IOCS.

SINIX transparent mode

SINIX transparent mode refers to the normal programming of printers on SINIX systems, e.g. from C programs. The C commands (e.g. *fprintf*) transfer a user buffer containing control sequences to the printer. These control sequences are not interpreted by the operating system, and must therefore be adapted to the respective printer. The application program does not receive any acknowledgements of errors.

Spool

The tasks of the spooler can be summarized under administrative aspects such as assigning and ordering

devices and managing queues, as well as under printer supported aspects such as page format and header pages. The spooler functions are, on the one hand, offered as shell commands (for printing files) and, on the other hand, as a procedural interface for integration in application programs.

IOCS

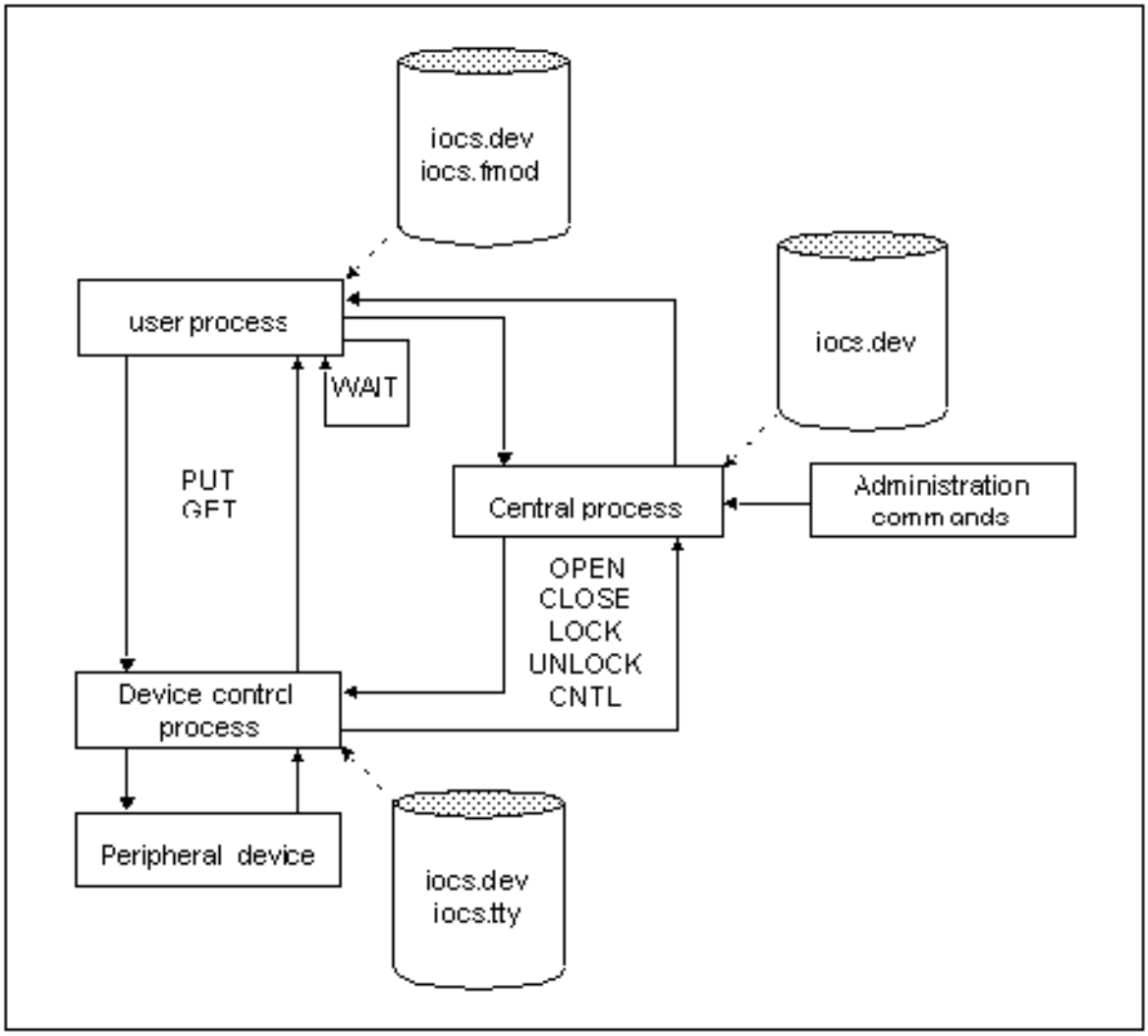
IOCS is the connecting layer between the commands and data of the application programs and the peripheral devices (see also Section "IOCS process concept"). It interprets function calls (in the IOCS user interface), converts the compatible control sequences to printer-specific sequences and provides a code conversion from internal system code to the respective target code of the printer. Moreover, it provides users with error- and status acknowledgements from the printer.

Workstations/printers

SAS and RS-232C printers can be connected to both intelligent workstations and terminals using the programming interfaces described above. In addition, line printers can be connected directly to the system and some RS-232C printers can also be connected directly to the system.

1.4 IOCS process concept

The following diagram illustrates the flow of communication between the application and peripheral devices.



Application process

The application process is the application program linked to an IOCS library via the IOCS user interface.

Central process

The central process of IOCS is a continuous administration process which exists only once on the system. It manages the devices to be operated by IOCS, and starts/stops the necessary device control processes (see Chapter "Administration of IOCS").

Device control process

The device control process is the IOCS output process. It transfers the IOCS commands (see Chapter "User interface for printers") and the compatible control sequences (see Chapter "Compatible control sequences") in the specific output formats (depending on the device) and controls one or more peripheral devices. OPEN, CLOSE, LOCK, UNLOCK, and CNTL requests are output from the central process to the device control process. The device control process receives PUT and GET requests directly from the application program. *pr_gsm1* and *pr_nd90* are provided as default device control processes for printers (see also Chapter "Configuration of IOCS"). In addition to the default device control processes, users can create their own (see the manual "Generating Device Control Processes").

Administration commands

The administration commands are used to start and stop the central process, to query the status of IOCS, to delete applications from the list of active applications, to activate and deactivate the process trace, and to activate a configuration change (see Chapter "Administration of IOCS").

Configuration files

The iocs.dev file

The *iocs.dev* file describes all devices and device groups to be operated via IOCS.

The iocs.fmod file

The *iocs.fmod* file describes the functionality of the available device control processes.

The iocs.tty file

The *iocs.tty* file describes the device-specific TTY parameters.

1.5 Operable printers

The following table gives an overview of the printers described in this manual which may be operated via IOCS:

Sales name	Alias	Sales name	Alias
HPR 4007-N10□	ND48□	9042□	ZD12□
HPR 4007-N60□	ND49□	9043□	ZD13□
HPR 4009-N10□	ND65□	9044□	ZD14□
HPR 4009-N60□	ND66□	MD06□	–□
HPR 4010-N10□	ND72□	MD14□	–□
HPR 4010-N60□	ND73□	ND24□	–□
HPR 4014-N60□	ND38□	ND25□	–□
HPR 4806-P10□	MD12□	ND27□	–□
HPR 4810-P10□	MD26□	ND31□	–□
HPR 4815-P10□	MD20□	ND33□	–□
HPR 4904□	ND90□	ND37□	–□
9014-12□	ND68□	ND44□	–□
9021□	MD24□	ND45□	–□
9022-200□	MD07	TD06□	–□
9022-200□	(HP-Laserjet)□	TD08□	–□
9041	MD07 (Diablo 630)□	ZD09	–

	ZD11		

1.6 Device types of operable printers

The following table gives an overview of the device types of the printers that can be operated via IOCS:

Printer	Device type
ND24, ND25, ND27, ND44, ND45, TD06, TD08	SIIPR
ND90	ND90PR
all other printers	with serial connection: V_24□ with parallel connection: LP or LP_C

1.7 Presentation of syntax

To help you find particular sections and to make the manual easier to use, the chapter and section headings are displayed in the running head.

The chapter heading appears on the outside of the running head, while the title of the respective section appears on the inside.

In order to make it easier to locate information, particular words are highlighted by their font.

The following conventions have been used in the body text:

italics	For path names, file names, command names, options, constants, and variables
typewritten text	for system output (e.g. messages) and examples

The following conventions have been used for syntax diagrams:

typewritten text	For programming instructions
bold	For path names, file names, command names, options, and constants
normal type	For variables
[□]	May be omitted; the brackets must not be entered
Ë	Mandatory blank
...	tPrevious expression may be repeated any number of times

2 Administration of IOCS

2.1 Preface

A range of commands is available for IOCS administration; these commands can be used to perform the following functions:

- Controlling the process traces
- Starting/stopping the central process
- Querying the IOCS status
- Deleting applications from the list of active applications

2.1.1 Purpose

This chapter describes the different administration functions.

2.1.2 Layout of section

The Section "Process trace" gives an overview of the process trace options. This administration function includes various process trace options for different processes.

The Section "Administration commands" describes the commands that underlie the administration functions.

2.2 Process trace

There are two different types of process trace:

- Abbreviated process trace
- Detailed process trace

Each of these two types can be activated for any of the following processes:

- IOCS application process
- Central process
- Device control process

The results of a process trace (PT) are stored in appropriate PT files, depending on the process:

Process	PT file
IOCS application process	*$IOCS_TRACE/lu.user_id.process_id*
Central process	*$IOCS_TRACE/logzp.process_id*
Device control process	*$IOCS_TRACE/lg.process_id*

Both PT types provide different results, which are reflected in the PT levels. Each level also supplies the results of its subordinate level(s); for example, level 3 also returns the results of levels 1 and 2.

Level	PT type	Target groups	In order to find	Results
0	no PT			
1	abbreviated PT	system administrators SNI Service	configuration errors	messages and internal errors
2	detailed PT for customer	system administratorsprogrammers	invalid sequences and characters	display of input and output data
3	detailed PT for service	SNI service	(internal) errors in the	display of table contents, device

			configuration, the devices, etc.	status, and device protocol
4	detailed PT for developer	developers	program errors	not specified
5	detailed PT for developer	developers	memory errors	not specified

The PT levels are defined differently when activating a process trace in the various processes (see Section "Process traces in the IOCS application process", Section "Process trace in the central process" and Process trace in the device control process).

2.2.1 Process traces in the IOCS application process

An abbreviated process trace can be switched on at the runtime of the process using the command *iocs_trace*. This process trace is only valid for the runtime of the process (at most).

To activate a detailed process trace, the corresponding application program is linked again to either of the two libraries, *libxio_t.a* or *libxio_t.so*, and then started. The default PT level is "3". However, this can be overridden by setting the shell variable *IOCS_LEVEL*.

There are two ways of switching off the process trace:

- The command *iocs_notrace* causes the process trace to be switched off for the runtime of the process.
- Long-term deactivation of the detailed process trace is achieved by linking the appropriate application to either of the two libraries *libxio.a* or *libxio.so* and then starting the application. In addition, the shell variable *IOCS_LEVL* must be set to "0" if it were set at activation.

2.2.2 Process trace in the central process

The abbreviated process trace can be switched on at the runtime of the process using the *iocs_trace* command, or it can be activated by using the *iocs_start* command and specifying a PT level of "1" when starting the process.

The detailed process trace can be switched on by using the *iocs_start* command and specifying a PT level other than "0" or "1" when starting the process.

The *iocs_notrace* command is used to switch off the process trace. When the central process is terminated using the *iocs_stop* command, the process trace is likewise switched off.

2.2.3 Process trace in the device control process

The abbreviated process trace can be switched on at the runtime of the process using the *iocs_trace* command. This trace is only valid for the runtime of the process (at most).

If a device control process is configured with the *.t* extension and a PT level of "1" in the *iocs.dev* file, the abbreviated process trace for this device control process is activated when the process is started. (The *.t* extension identifies a device control process as having a process trace.)

If a device control process is configured without the *.t* extension, but with a PT level within the range "1" to "5" in the *iocs.dev* file, the abbreviated process trace for this device control process is likewise activated when the process is started.

If a device control process is configured with the *.t* extension and possibly with a PT level other than "0" or "1" in the *iocs.dev* file, the detailed process trace for this device control process is activated when the process is started. If an invalid or no PT level is specified, level "3" is assumed automatically.

The PT level can be changed in the *iocs.dev* file, without having to explicitly activate the change. The next time the relevant device control process is started, the change is activated automatically.

There are two ways of deactivating the process trace:

- The process trace can be switched off for the runtime of the active process using the *iocs_notrace* command.
- A long-term deactivation of the detailed process trace is achieved by configuring the relevant device control process in the *iocs.dev* file without the *.t* extension.
- The abbreviated process trace can be deactivated on a long-term basis by deleting the PT level entry for the appropriate device control process in the *iocs.dev* file.

2.3 Administration commands

2.3.1 Starting the central process

Synopsis

iocs_startË[PT level]

Description

This command firstly creates the directories required by IOCS, namely *$IOCS/fifos*, *$IOCS/tmp/iocs*, and *$IOCS_TRACE* (provided they do not already exist).

By default, *$IOCS_TRACE* is *$IOCS/trace*. The user has the option of assigning another path to the *$IOCS_TRACE* variable. In this case, a path *directory_name/trace* must be entered instead of *$IOCS/trace* in the */etc/profile* file. This amendment must be made before *iocs_start* is invoked, and *directory_name* must exist.

The central process is then started. The central process checks whether a central process has already been started. If this is the case, the newly started process issues an error message and terminates itself. If no error is found, the central process sets up its communication channel and waits for requests.

Specification of the *PT level* parameter activates the process trace in the central process to be started. Depending on the value of the entry, the following PT types are started:

Value	PT type
1	abbreviated PT
2	detailed PT (level 2)
3	detailed PT (level 3)
4	detailed PT (level 4)
5	detailed PT (level 5)
>5	detailed PT (highest level)
invalid entry	detailed PT (level 3)

If no PT level is specified, or if "E0" is entered, the central process is started without a process trace.

Errors

Cannot open trace file *filename*, <errno> *error_no*

Cause:
The PT file could not be opened.

Reaction:
Check that the directory exists, and check the access rights.

Cannot create communication file $IOCS/fifos/*filename*, <errno >*error_no*

Cause:
The communication file could not be created.

Reaction:
Check that the directory exists, and check the access rights.

Cannot open communication file $IOCS/fifos/*filename*, <errno> *error_no*

Cause:
The communication file could not be opened.

Reaction:
Check that the directory exists, and check the access rights.

Cannot open/read configuration file *directory*/lib/iocs.dev, <errno> *error_no*

Cause:
The configuration file *iocs.dev* could not be opened or read.

Reaction:
Check that the directory exists, and check the access rights.

GSP name *gsp_name* too long

Cause:
The name of the device control process exceeds 256 bytes.

Reaction:
Select a shorter name.

IOCS is already running

Cause:
A central process has already been started.

Note

The central process is started automatically when the system is powered up.

2.3.2 Stopping the central process

Synopsis

iocs_stop

Description

This command sends a message to the central process, which then stops all device control processes and sends the message *E_XIOSTOP* to all applications. Finally, the central process terminates itself.

Errors

IOCS not started

Cause:
No central process was started.

IOCS has been stopped

Cause:
The active central process was stopped (status message).

Note

All applications must be terminated before issuing this command, as otherwise they will have to be restarted.

2.3.3 Querying the IOCS status

Synopsis

iocs_status

Description

This command sends a message to the central process, which then outputs the current status of the IOCS runtime environment to the screen in the following form:

```
IOCS Rel. 3.0
STATUS :   3 user
UIPD : 68064
DEV  GRP  PID-GSP   STATUS
4    2    68096    LOCKED
5    3    68031    UNLOCKED
UIPD : 68078
DEV  GRP  PID-GSP   STATUS
5    3    68031    UNLOCKED
UIPD : 68022
DEV  GRP  PID-GSP   STATUS
4    2    68096    UNLOCKED
```

Errors

IOCS not started

Cause:

No central process was started.

iocs_zp returns error *error_no*

Cause:

Error reported by the central process.

Reaction:

Inform the SNI Service Center.

Error in process communication

Cause:

Error in communication between the IOCS processes.

1. The *$IOCS/fifos/filename* file could not be opened. The directory or file does not exist, or has the wrong access rights.
2. Error when reading/writing the file. The process is no longer active.

Reaction:

1. Check that the directory/file exists, and check the access rights.
2. Determine the cause of the crash.

2.3.4 Deleting applications from the list of active applications

Synopsis

iocs_canËprocess-IDË...

Description

This command informs the central process that one or more applications are to be deleted from the list of active applications. The process IDs of the applications are transferred to the central process as parameters. The central process then generates the necessary UNLOCK and CLOSE requests for the devices opened by these applications, so that they can be closed properly.

Errors

IOCS not started

Cause:
No central process was started.

Illegal PID *process_id*

Cause:
The specified process ID is invalid.

Reaction:
Check the process ID.

iocs_zp returns error *error_no*

Cause:
Error reported by the central process.

Reaction:
Inform the SNI Service Center.

Process *process_id* is not known in IOCS

Cause:
IOCS does not recognize the process.

Reaction:
Check the process ID.

Error in process communication

Cause:
Error in communication between the IOCS processes.

1. The *$IOCS/fifos/filename* file could not be opened. The directory or file does not exist, or has the wrong access rights.
2. Error when reading/writing the file. The process is no longer active.

Reaction:

1. Check that the directory/file exists, and check the access rights.
2. Determine the cause of the crash.

Cannot get memory

Cause:
There is no more memory available.

Reaction:
Increase system resources.

Application *programm_id* is cancelled

Cause:
The specified application was terminated (status message).

2.3.5 Activating the abbreviated process trace

Synopsis

iocs_traceËprocess-IDË...

Description

This command activates the abbreviated process trace for the specified processes.

If one of the specified traces is inactive, or has been omitted from the IOCS as a result of an operator error

(e.g. wrong process ID specified), an appropriate error message is issued. However, the command is executed for all other processes specified.

Errors

IOCS not started

Cause:
No central process was started.

Illegal PID *process_id*

Cause:
The specified process ID is invalid.

Reaction:
Check the process ID.

iocs_zp returns error *error_no*

Cause:
Error reported by the central process.

Reaction:
Inform the SNI Service Center.

Process *process_id* is not known in IOCS

Cause:
IOCS does not recognize the process.

Reaction:
Check the process ID.

Cannot get memory

Cause:
There is no more memory available.

Reaction:
Increase the system resources.

Error in process communication

Cause:
Error in communication between the IOCS processes.

1. The *$IOCS/fifos/filename* file could not be opened. The directory or file does not exist, or has the wrong access rights.
2. Error when reading/writing the file. The process is no longer active.

Reaction:

1. Check that the directory/file exists, and check the access rights.
2. Determine the cause of the crash.

Tracing is already active for process *process_id*

Cause:
The process trace is already active for this process.

Reaction:
Check the process ID.

Release of process *process_id* is less than V3.0

Cause:
The release of the IOCS process is lower than version 3.0.

Reaction:
The application must be linked again to either of the libraries */usr/lib/libxio_t.a* or *libxio_t.so* for a process trace in the IOCS application process. When the application is restarted, a PT file called *lu.user_id.process_id* is created in the *$IOCS/tmp/iocs* directory.

For process traces in the device control process, the corresponding device control process must be entered in the *iocs.dev* file with the *.t* extension, and the central process must be stopped and restarted. After starting the central process, a PT file called *lg.process_id* is created for the relevant device in the *$IOCS/tmp/iocs* directory.

Tracing for process *process_id* will be switched off

Cause:
The process trace is activated for the specified process (status message).

2.3.6 Deactivating the process trace for the duration of the process runtime

Synopsis

iocs_notraceËprocess-IDË...

Description

This command deactivates the process trace for the entire runtime of the specified process.

If one of the specified processes is inactive, or is omitted from the IOCS as a result of an operator error (e.g. wrong process ID specified), an appropriate error message is issued. However, the command is executed for all other processes specified.

Errors

IOCS not started

Cause:
No central process was started.

Illegal PID *process_id*

Cause:
The specified process ID is invalid.

Reaction:
Check the process ID.

iocs_zp returns error *error_no*

Cause:
Error reported by the central process.

Reaction:
Inform the SNI Service Center.

Process *process_id* is not known in IOCS

Cause:
IOCS does not recognize the process.

Reaction:
Check the process ID.

Cannot get memory

Cause:

There is no more memory available.

Reaction:

Increase the system resources.

Error in process communication

Cause:

Error in communication between the IOCS processes.

1. The *$IOCS/fifos/filename* file could not be opened. The directory or file does not exist, or has the wrong access rights.
2. Error when reading/writing the file. The process is no longer active.

Reaction:

1. Check that the directory/file exists, and check the access rights.
2. Determine the cause of the crash.

Tracing isn't active for process *process_id*

Cause:

The process trace is inactive for this process.

Reaction:

Check the process ID.

Release of process *process_id* is less than V3.0

Cause:

The release of the IOCS process is lower than version 3.0.

Reaction:

To switch off the process trace in the IOCS application process, the application must be linked again to either of the libraries *libxio.a* or *libxio.so*.

To switch off the process trace in the device control process, the corresponding device control process must be entered in the *iocs.dev* file without the *.t* extension, and the central process must be stopped and restarted.

Tracing for process *process_id* will be switched off

Cause:

The process trace is deactivated for the specified process (status message).

2.3.7 Activating configuration changes

Synopsis

iocs_configË–[c]r

Description

Using this command, the following configuration changes entered in the *iocs_dev* file are activated for the current central process:

- Insertion of a device group, including the devices
- Deletion of a device group, including the devices, provided no applications are currently working with this device group
- Modification of the device control process names (e.g. in order to activate the process trace)

If the *-c* option is used, the (modified) configuration is checked for consistency. Any errors found are output to the screen. The following tests are carried out:

- Uniqueness of group numbers, device numbers, and path names.
- Existence of the device control processes entered under *$IOCS/bin* or of their absolute paths and specifications in the *iocs.fmod* file.
- Consistency between the files *iocs.dev* and *iocs.fmod*.
- Existence of the configured tables, and uniqueness of table numbers in the *iocs_dev* file in the case of default device control processes.
- Existence of the emulation table for the device types *V_24* and *LP*, in the case of default device control processes.

Errors

IOCS not started

Cause:
No central process was started.

iocs_zp returns error *error_no*

Cause:
Error reported by central process.

Reaction:
Inform the SNI Service Center.

Cannot get memory

Cause:
There is no more memory available.

Reaction:
Increase system resources.

Error in process communication

Cause:
Error in communication between the IOCS processes.

1. The *$IOCS/fifos/filename* file could not be opened. The directory or file does not exist, or has the wrong access rights.
2. Error when reading/writing the file. The process is no longer active.

Reaction:

1. Check that the directory/file exists, and check the access rights.
2. Determine the cause of the crash.

Inconsistent new configuration

Cause:
The new IOCS configuration is inconsistent.

Reaction:
Check the configuration.

New configuration is available

Cause:
The new configuration is available (status message).

3 Configuration of IOCS

3.1 Preface

In order to manage devices using IOCS, they must be configured in the system.

These devices are configured either with the help of the Config tool, which can be called up using SINIX system administration (SYSADM) (see the manual "Character User Interface for SINIX System Administration"), or "manually", by creating the corresponding printer nodes using SINIX commands and by editing and modifying the configuration files.

3.1.1 Purpose

This section of the manual describes the configuration of devices to be managed using IOCS.

3.1.2 Layout of section

The Section "First configuration" describes the procedures for configuring IOCS on your system for the first time.

The Section "Reconfiguration" describes the procedures for changing the configuration of an IOCS already configured on your system.

Section "Description of the iocs.dev configuration file", Section "Description of the iocs.fmod configuration file" and Description of the iocs.tty configuration file describe the structure of the files *iocs.dev*, *iocs.fmod*, and *iocs.tty*.

3.2 First configuration

When configuring IOCS using the Config tool, the printer nodes are created in the system and all necessary entries are made automatically in the configuration files. The configuration changes are activated when Config is exited by choosing *set* in the *quit* menu.

If there is no Config tool on the system, IOCS can also be configured "manually".

In this case, the corresponding printer nodes must be created on the system using *termadd(1M)* or *lpsetup(8)*, and the corresponding entries must be made directly in the configuration files. Correct entries for the *iocs.dev* file can be taken from the *iocs.dflt* file. The user should note that the pathnames of the devices must still be adapted.

When configuring "manually" for the first time, the planned configuration is activated by starting the central process.

There are certain constraints to be noted when using Config:

- Only printers offered in the selection list can be specified. This means on the one hand that some printers are offered which cannot be operated via IOCS and, on the other hand, that not all of the printers that can be operated via IOCS appear on the list.

  If printers which do not appear on the list are to be configured, they must be configured "manually".
- TTY parameters cannot be modified.

  If TTY parameters are to be modified (e.g. the baud rate), the corresponding entries must be made directly in the *iocs.tty* file.
- No entries are made in the configuration files for device control processes created by the user or device control processes with detailed process traces.

  Entries for device control processes created by the user and for detailed process traces in device control processes must be made directly in the *iocs.dev* or *iocs.fmod* file.

3.3 Reconfiguration

Changes to an existing configuration can be made either with the Config tool or "manually".

When using Config, the active central process must firstly be stopped (if no IOCS application is running, the

central process is stopped automatically by Config). The configuration changes are activated when Config is exited by choosing *set* in the *quit* menu. The same constraints apply as described in the Section "First configuration".

When changing a configuration "manually", the central process need only be stopped if device numbers and/or group numbers are to be changed in the *iocs.dev* file. All other changes can be made while the central process is running, and can be activated using the *iocs_config* command (see the Chapter "Administration of IOCS").

3.4 Description of the iocs.dev configuration file

All devices to be operated via IOCS are described in the *iocs.dev* file. Here, the user can enter device groups and devices, and can specify code conversion table and emulation tables, as well as owner and access rights for individual devices.

The file consists of modules which are written on a line-by-line basis. The first item written to a module is always a keyword, and the data ends with the end of the line. Lines which do not begin with a keyword are interpreted as comment lines. Parameters within a description line are separated by blanks. The *iocs.dev* file consists of the following modules:

3.4.1 Device groups

Synopsis

$DEVGËgroup_numberËdevice_control_process[**.t**]Ë[PT_level]

Description

Devices which are to be operated via IOCS are formed into groups in a group description. Each device group comprises at least one device. The maximum number of devices per group is defined in the *iocs.fmod* file. In the case of printers, each group comprises one printer only.

$DEVG indicates a group description line.

In the *group_number* parameter, the group is assigned a serial number, which must be unique in the file and lie within the range of values "1" to *IO_GRPMAX* (inclusive). The constant *IO_GRPMAX* is defined in the *iocs_config.h* file.

The name of a device control process is specified in the *device_control_process* parameter. The default device control processes *pr_nd90* (for the ND90) and *pr_gsm1* (for all other printers) are available for printers. The *.t* extension identifies a device control process as having a process trace.

The level for the process trace in the device control process can be specified in the *PT_level* parameter (see also the Section "Process trace" in the Chapter "Administration of IOCS"). Depending on the value of this parameter and on the existence of the *t* extension in the device control process name, the following PT types are started:

Value	PT type(name with extension)	PT type(name without extension)
0	no PT	no PT
1	abbreviated PT	abbreviated PT
2	detailed PT (level 2)	abbreviated PT
3	detailed PT (level 3)	abbreviated PT
4	detailed PT (level 4)	abbreviated PT
5	detailed PT (level 5)	abbreviated PT
>5	detailed PT (level 5)	abbreviated PT
invalid specification	detailed PT (level 3)	abbreviated PT

no specification	detailed PT (level 3)	no PT

3.4.2 Devices

Synopsis

$DEVËdevice_numberËpathnameËdevice_typeË[device_type]

Description

Following the group description, the devices to be assigned to these groups are specified on individual lines. All devices up to the next group description thus belong to this group.

$DEV indicates a device description line.

In the *device number* parameter, a device is assigned a serial number, which must be unique in the file and must lie within the range of values "1" to *IO_DEVNCNT* (inclusive). The constant *IO_DEVNCNT* is defined in the *iocs_config.h* file.

The absolute path name of a device is specified in the *pathname* parameter.

The driver of the respective printer is specified in the *device type* parameter. The assignment of individual printers to a device type can be found in the list of connectable printers in the introduction to this manual.

The precise device type can be specified in the *device type* parameter. This specification is optional, apart from two exceptions. For the ND90 printer, *ND90* must be specified, and *MD07D630* must be specified for the 9022-200 printer with *Diablo 630* emulation.

3.4.3 Data tables

Synopsis

$DTABËtable_nameËtable_number

Description

Data tables for code conversion can be specified in individual lines for each device. Table descriptions are given for each individual device, i.e. the table descriptions must directly follow the description line of the particular device. Data tables can be specified for all printers. Up to *IO_TBMAX* data tables are permitted per device. The *IO_TBMAX* constant is defined in the *iocs_config.h* file. If no data table is specified, the device operates without code conversion.

$DTAB indicates a description line for data tables.

The name of a data table for code conversion is specified in the *table_name* parameter. The specified table must exist in the *$IOCS/tables* directory. Names of data tables always end with the *.iocs* extension.

In the *table_number* parameter, the data table is assigned a serial number, which must be unique for the relevant device.

3.4.4 Emulation tables

Synopsis

$ETABËtable_name

Description

Specification of an emulation table is only permitted for the device types *V_24*, *LP*, and *LP_C*. One emulation table must be specified for each device. Table descriptions are made for each individual device, i.e. the table descriptions must directly follow the description line of the relevant device.

$ETAB indicates a description line for emulation tables.

The name of an emulation table for converting control sequences is specified in the *table_name* parameter. The table specified must exist in the *$IOCS/tables* directory. Names of emulation tables always have the *.cote* extension.

3.4.5 Owner and access rights

Synopsis

$ACCËaccess_rightsËuser_IDËgroup_ID

Description

Owner and access rights can be specified in one line for each device. These rights are created for each individual device, i.e. their description must follow the relevant device description, as with tables. If the owner and access rights are not described, the user ID and group ID of IOCS are assumed, and the device is made available to all users for direct printing.

$ACC indicates a description line for owner and access rights.

The access rights for the user group "Owner Group Other" are specified in the *access_rights* parameter. An "x" means that the device is available to the corresponding user group for direct printing; an "–" means that the device is locked.

The owner rights for the device are specified in the *user_ID* and *group_ID* parameters.

3.4.6 Sample entries in the iocs.dev file

```
$DEVG   1   pr_gsm1
$DEV   1   /dev/tty22p   SIIPR
$DTAB   sas2.iocs    1
$DEVG   2   pr_gsm1
$DEV   2   /dev/tty26q   V_24    MD07D630
$DTAB   md07dia.iocs  1
$ETAB   md07dia.cote
$DEVG   3   pr_gsm1
$DEV   3   /dev/tty49q   V_24    MD07HPLJ
$DTAB   9022.iocs    1
$ETAB   9022.cote
$DEVG   4   pr_gsm1
$DEV   4   /dev/lp1     LP
$ETAB   linetab.cote
$DEVG   5   pr_nd90
$DEV   5   /dev/tty10p   ND90PR  ND90
$DTAB   nd90.iocs    1
```

3.5 Description of the iocs.fmod configuration file

In the *iocs.fmod* file, the function range of the existing device control processes is described. Here, users can define the functionality of device control processes they have created themselves. The function range of the default device control processes is already defined in this file in the factory.

The file consists of modules which are written on a line-by-line basis. The first item written to a module is always a keyword, and the data ends with the end of the line. Lines which do not begin with a keyword are interpreted as comment lines. Parameters within a description line are separated by blanks. The *iocs.fmod* file comprises the following modules (see the following page):

3.5.1 Device control process

Synopsis

$MODËGSP_nameË[no_of_devices]

Description

$MOD indicates a description line of a device control process.

The name of the device control process to be described is specified in the *GSP_name* parameter. The name must have the same syntax as in the *iocs.dev* file.

The number of devices that can be operated simultaneously by this device control process is specified in the *no_of_devices* parameter. If no value is specified, the value "1" is adopted by default. Default device control

processes for printers can each only operate one device.

3.5.2 Function modules

Synopsis

$FMËmodule_numberËmodule_nameË[no_of_jobs]Ë[operating_mode]

Description

Following the description line of the device control process, the function modules integrated in this device control process are described on individual lines.

$FM indicates a description line of a function module.

In the *module_number* parameter, the function module is assigned a serial number, which must be unique within the device control process description.

The name of the function module is specified in the *module_name* parameter. The name can normally be freely selected, except when printing via the spooler, in which case *PRINTMOD* is mandatory. The module names *EPSMOD* and *DEVMOD*, which are possibly entered for default device control processes, are still available for compatibility reasons.

The maximum sequence number (permitted number of jobs before a WAIT command is issued) is specified in the *no_of_jobs* parameter. If there is no specification, the value "1" is assumed by default. In order to keep resources free for other processes, only a relatively low value ("2" or "3") should be chosen.

In the *operating_mode* parameter, it is possible to define whether an application is to work exclusively with this device control process (*EXCL HALFD*), or whether multiuser operation is possible (NOEX HALFD). If there is no specification, *NOEX HALFD* is assumed by default.

3.5.3 Device types

Synopsis

$SVRËdevice_type

Description

Following the description of the function modules, the device drivers integrated in this device control process are described on individual lines.

The following values can be specified for printers in the *device_type* parameter:

V_24 for printers with a serial connection
LP, LP_C for printers with a parallel connection
SIIPR for printers with an SASII connection
ND90PR for the ND90

3.5.4 Sample entries in the iocs.fmod file

```
$MOD   pr_gsm1     1
$FM   1   PRINTMOD   2   NOEX HALFD
$SVR   V_24
$SVR   SASIIPR
$SVR   LP
$SVR   LP_C
$MOD   pr_nd90     1
$FM   1   PRINTMOD   3   EXCL HALFD
$SVR   ND90PR
```

3.6 Description of the iocs.tty configuration file

Synopsis

pathname[:TTY_parameters][:special_string]

Description

Device-specific parameters for devices connected directly to the system are defined in the *iocs.tty* file. The existence of this file, and an entry for a corresponding device, are optional. Each data entry consists of one line, whereby the individual fields are separated by a colon.

The path name of a device is specified in the *pathname* parameter. If the path name begins with */dev/term*, the relative path name can be specified; otherwise, the absolute path name must be specified.

TTY parameters can be set in the *TTY_parameters* parameter; as regards syntax and function, these parameters correspond to the modes described for the *stty(1)* command.

- In the case of default device control processes, the following parameter settings are generally used (for the names and meanings of the fields see *termio(7)*):

Field	Value
c_iflag	IXON | IXOFF
c_cflag	c_cflag & ~CLOCAL
c_oflag	0
c_lflag	0
c_cc[VMIN]	6 (or 2 for ND90 and MD07D630)
c_cc[VTIME]	1
c_cc[VQUIT]	28

  This means that only the physical parameters (baud rate, parity, and number of stop bits/data bits) can be set. All other fields remain unchanged.

- For device control processes created by the user, the number of parameters that can be changed depends on the programming of the device control process.

In the *special_string* parameter, the "Robust XON" function can be used with some directly connected printers for the default device control processes *pr_gsm1* and *pr_gsm1.t* by specifying *rob-xon wait_time*. with some directly connected printers. In this case, the device control process waits for the XON character for the period specified in *wait_time*. If the XON character does not arrive, the error *E_NOOPEN* is returned and the error *E_DEVNOOP* is output to the logging file. The XON character does not arrive if the printer is not switched on, if robust XON is not supported, or if it is not connected directly.

In the case of a device control process created by the user, default settings can be used in this parameter, depending on the programming (e.g. format settings for printers, or selection of the software protocol).

4 User interface for printers

4.1 Preface

The IOCS user interface was developed for programming printers and other workstation peripherals such as ID devices and readers. Programming of the various peripherals is almost identical due to the use of special commands.

4.1.1 Purpose

This part of the manual describes in detail how to program printers with the aid of IOCS.

4.1.2 Layout of section

The Section "General information on programming" explains the procedures involved in programming printers using the interface described in this manual, and the programming rules that apply.

The Section "C programming interface" describes the individual functions and includes programming examples.

The Section "Special control sequences for the ND90" lists non-compatible control sequences that are specific to the ND90.

The Section "Error messages and user reactions" describes specific user actions to the "defined error messages" and the "device driver error codes".

4.2 General information on programming

4.2.1 IOCS user interface

The IOCS user interface offers the following advantages thanks to the almost identical programming of various peripherals:

- LOCK/UNLOCK allow devices to be reserved for exclusive use by an application for a specific period of time.
- axplicit use of the WAIT command allows other routines in the program to run while the peripheral commands are executed (asynchronous programming). Under certain circumstances, this can produce a considerable improvement in processing speed.
- CNTL is used to obtain status information to which the program can respond.
- Whenever a command is issued, the devices return error codes, thus allowing the application to react appropriately.
- The use of ESC/P control sequences guarantees compatible programming of different types of printer. This applies to all printers approved for use with IOCS by Siemens Nixdorf Informationssysteme AG.
- Individual functions of printers can be implemented by addressing them using "native sequences", which IOCS transfers directly to the printer without prior conversion.
- Internal codes that can be used on SINIX systems (ISO 8859/x) are converted to different target printer codes, thus automatically taking country-specific variants into account. Code conversion by the IOCS can be switched off on intelligent workstations if conversion is to be performed at the workstation.

The IOCS user interface supports the following commands:

OPEN
: Opens and resets the printer

CLOSE
: Closes the printer

LOCK
: Locks the printer for other applications

UNLOCK

Unlocks the printer

PUT
: Transmits data and print control sequences

GET
: Reads data on printers equipped with reading devices

CNTL
: Reads device identification and status messages

: Cancels print jobs and sets timeouts for print jobs

WAIT
: Waits for the end of one or more device operations

These commands are described in this part of the manual. The section also includes examples showing how to use the commands. In order to use the IOCS interface, the relevant printer must be generated as an "IOCS printer" (please refer to the Chapter "Configuration of IOCS").

4.2.2 Control sequences

The PUT command is particularly important if you wish to make full use of the printer's special features. The command allows control sequences to be sent to the printer (in addition to the actual data being printed), thus initiating an appropriate printer reaction (e.g. paper feed). The Chapter "Compatible control sequences" describes the set of compatible control sequences supported, in order to facilitate compatible programming despite the variety of possible printers.

4.2.3 Page-based programming

Page-based programming should be used to ensure optimum throughput, i.e. all control characters on a print page should be sent to the printer with a single PUT command. This keeps the need for operating system control and administration routines to a minimum.

4.2.4 Error handling

If programming is page-based as described in the previous paragraph, the entire page (including all settings) must be repeated if an error occurs. This ensures that processing restarts correctly.

4.2.5 Routine processing

IOCS supports both asynchronous and synchronous routine processing for every device. The type of processing is selected using a flag in the OPEN command. In accordance with the mode selected, the necessary WAIT commands are executed automatically by the IOCS functions, and any error messages are transferred directly by the corresponding IOCS function.

The individual functions for asynchronous command processing are described in the ..... If a device is operated synchronously, only the procedure for error and status messages is changed.

4.2.5.1 Asynchronous routine processing

A device opened with the X_NOSYNC flag is operated asynchronously. In this case, the application must issue the necessary WAIT commands itself. The error messages produced by the individual functions are reported as return values. The device errors can be queried after the WAIT command in the *xio_wt* structure transferred. The device status messages are also made available to the application with the aid of this structure.

With PUT commands it is also possible to use alternate buffers in this mode, in order to improve the performance of devices. Moreover, the application can time monitor the execution of routines, and can cancel routines. If an application operates several devices asynchronously using the ICOS, a "multiple wait" can be used to wait for acknowledgments from several devices.

4.2.5.2 Synchronous routine processing

If a device is opened with the X_SYNC flag, all routines are handled synchronously. The necessary WAIT commands are executed implicitly by the corresponding functions. In this case, an error is generally reported

to the user as a return value of the function. If no errors occur while a routine is being processed, but the status has changed, the return value *E_NEWST* is transferred to the application. In this case the device status can be determined using the external pointer *xiopstat* from the structure *xio_prstat* or *xio_stnd90* for the ND90.

If a device is operated synchronously, the application cannot make use of alternate buffers, and is not able to time monitor or cancel routines. Moreover, a "multiple wait" cannot be carried out with this device.

4.2.6 Include files

The programming of individual functions, particularly the PUT command, is illustrated in the context of each definition with examples defined as external subcommands. The two files supplied, *xio.h* and *xio_desc.h*, must be linked in using the *#include* statement.

The *xio.h* file contains definitions of abbreviations and structures used by IOCS. The *xio_desc.h* file contains definitions of the escape sequences in the the ESC/P-NX/P compatibility box.

The *libxio.so*, *libxio_t.so*, *libxio.a* or *libxio_t.a* library must be linked in by the application at compile time. These librarys contains the various IOCS functions. The *xio_struct.h* file used in the examples contains the following definitions of structures and variables:

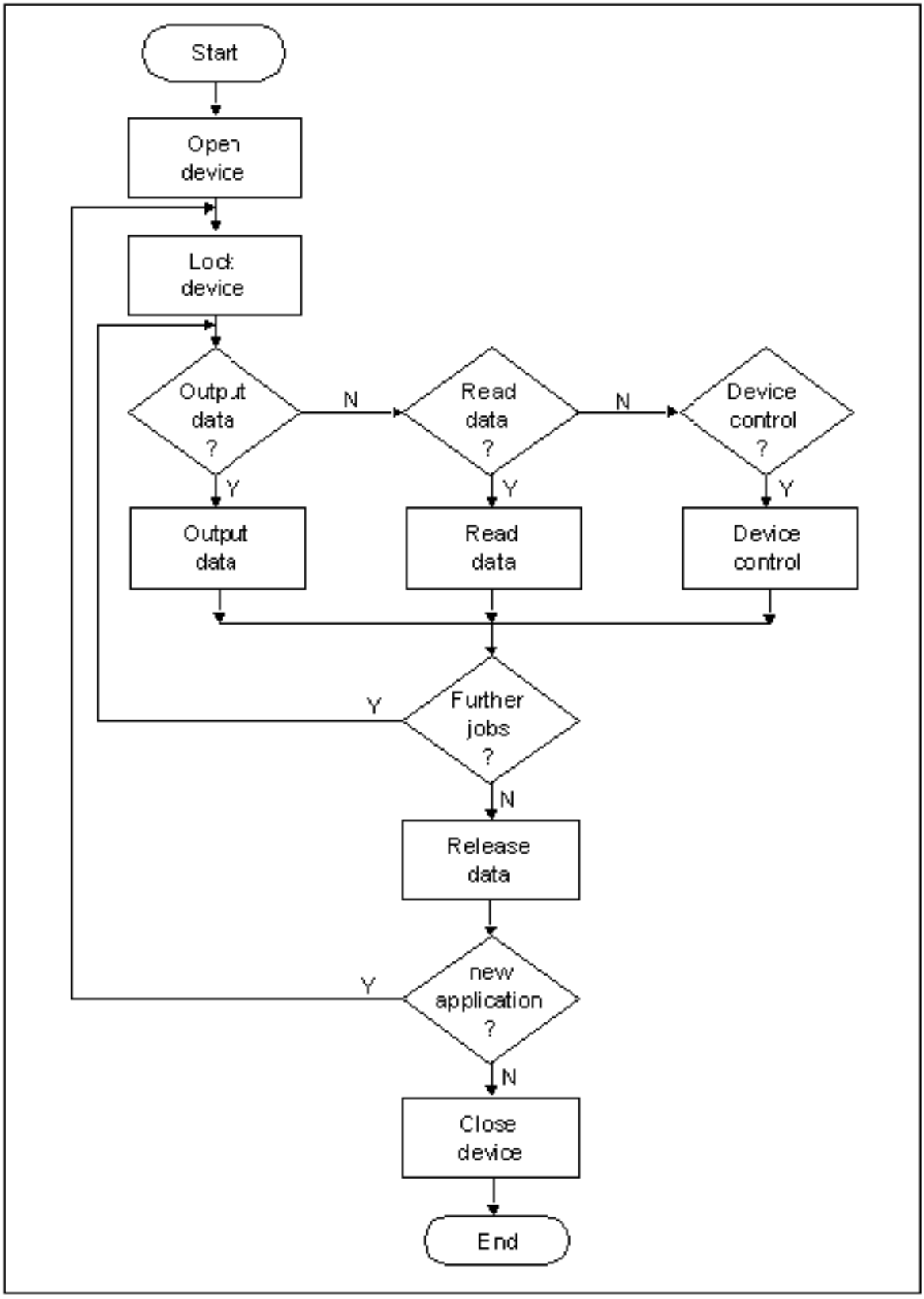
```
struct xio_ctl    checkup;   /* check structure (iocs_put/iocs_get)    */
struct prbl_data  line;      /* buffer for read data (iocs_get)        */
struct xio_idnd90 nd90ident; /* identification structure (iocs_cntl)   */
struct xio_stnd90 nd90stat;  /* status structure (iocs_cntl)           */
struct xio_arg    argument;  /* argument structure (iocs_cntl)         */
struct xio_wt     order;     /* command structure (iocs_wait)          */
union  xio_fct    function;  /* function structure (iocs_put/iocs_get) */
int    lastsequence;        /* (iocs_put)                             */
int    number = 0;          /* (iocs_put)                             */
int    maxsequence;         /* (iocs_open/iocs_put)                   */
```

4.2.7 Flowchart for print job processing

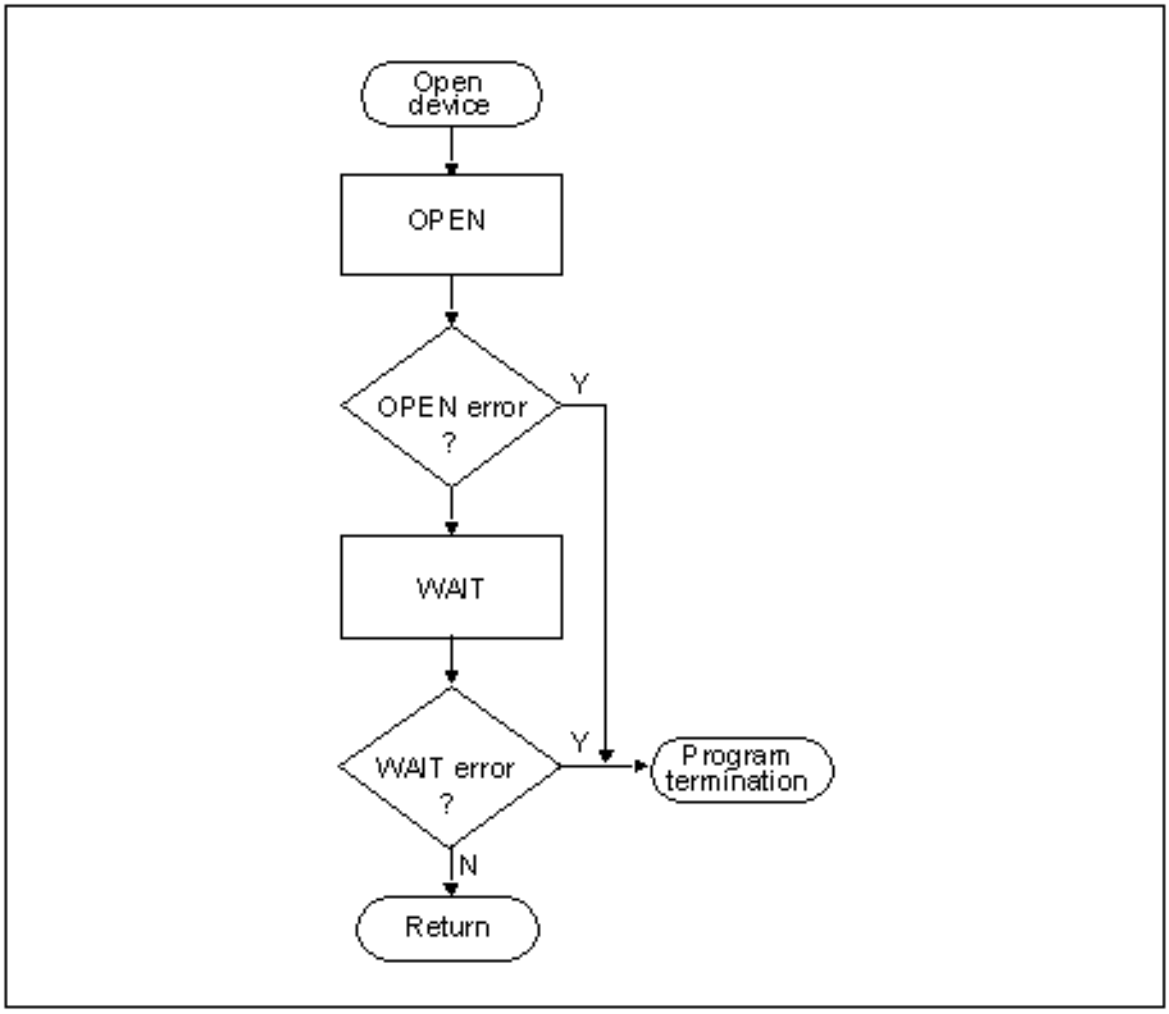
The flowchart on the following pages should help you when programming printers using IOCS.

It contains suggestions for implementing asynchronous print job processing, and clarifies the relationships between the individual IOCS commands and the optimum order in which they can be issued.

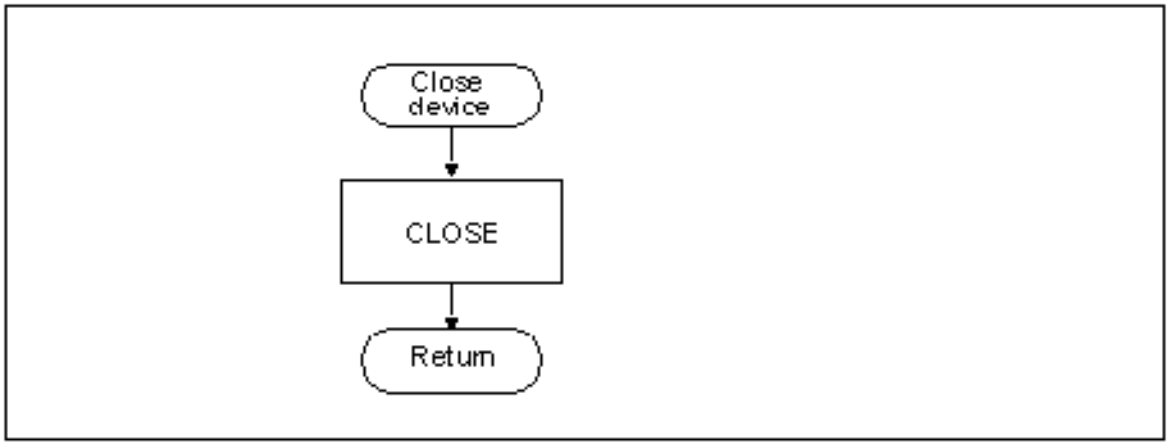
4.2.7.1 Control table



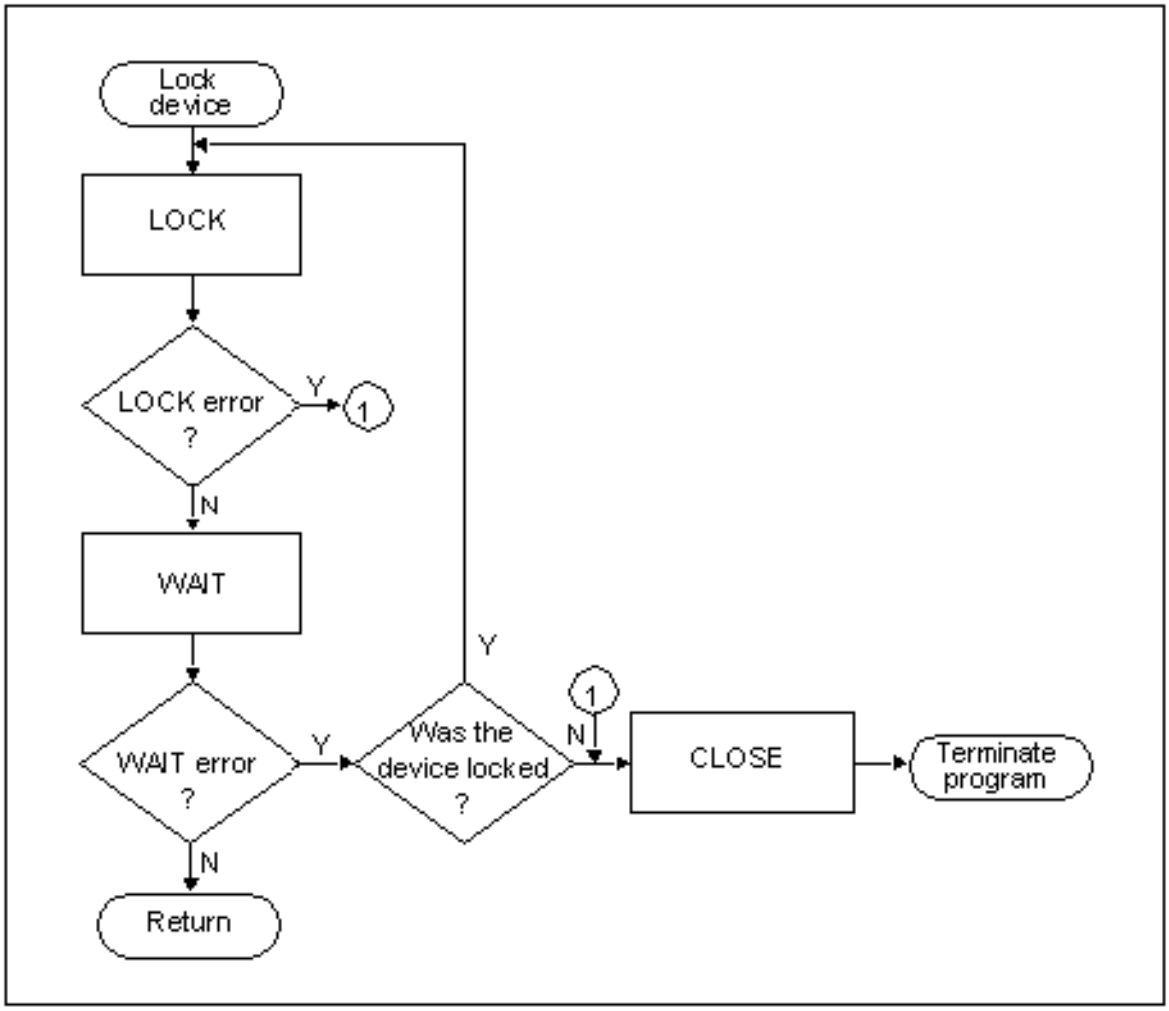
4.2.7.2 Open device



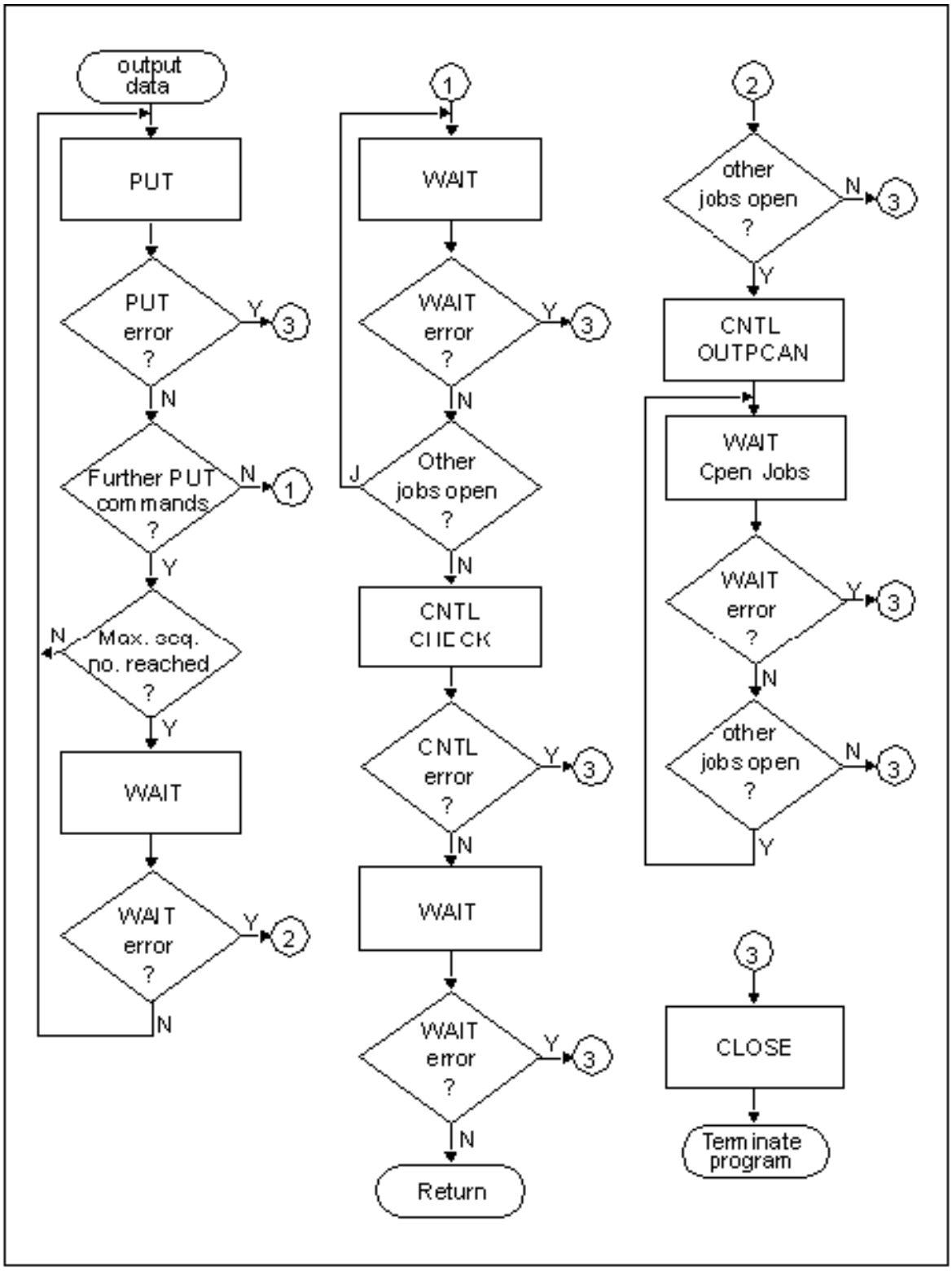
4.2.7.3 Close device



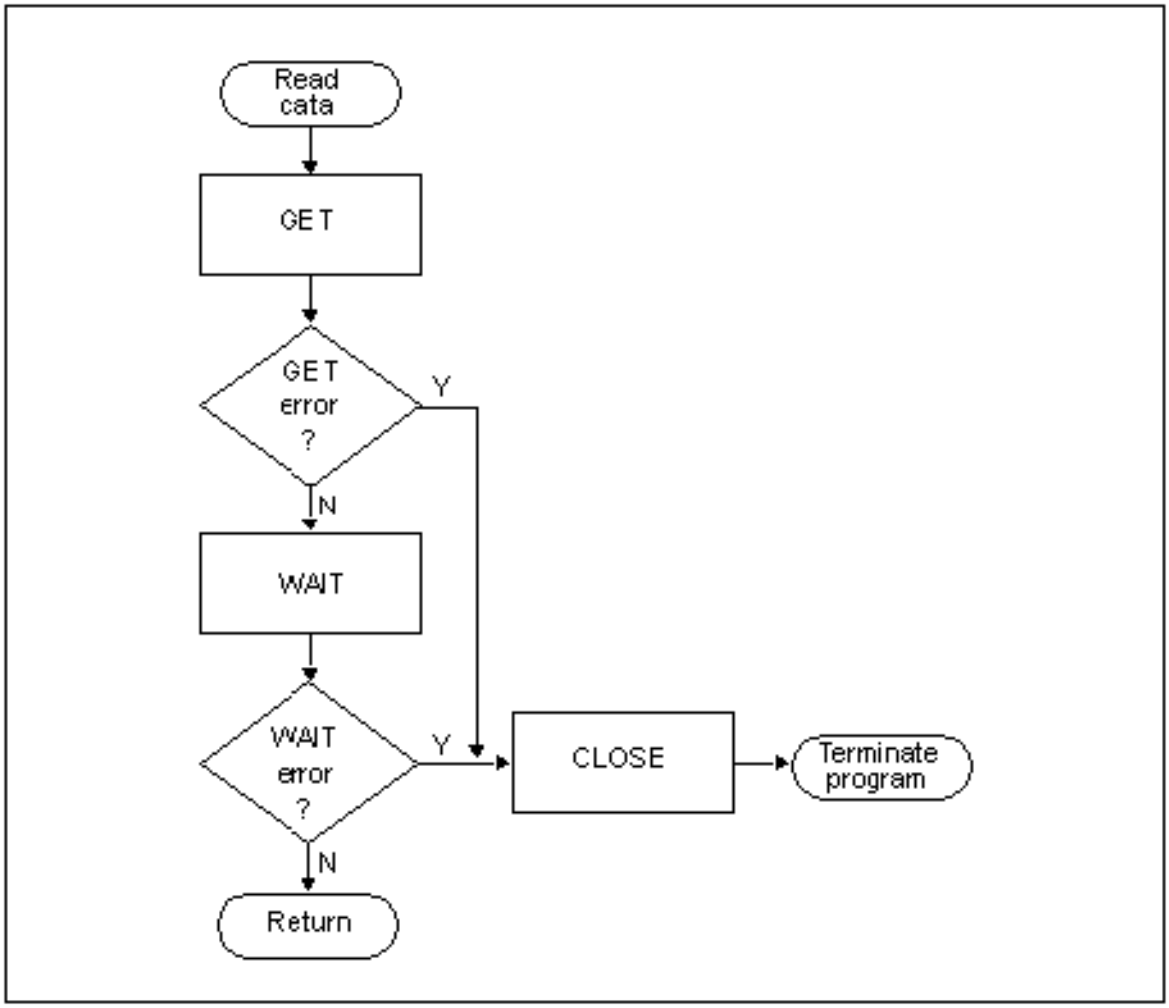
4.2.7.4 Lock device



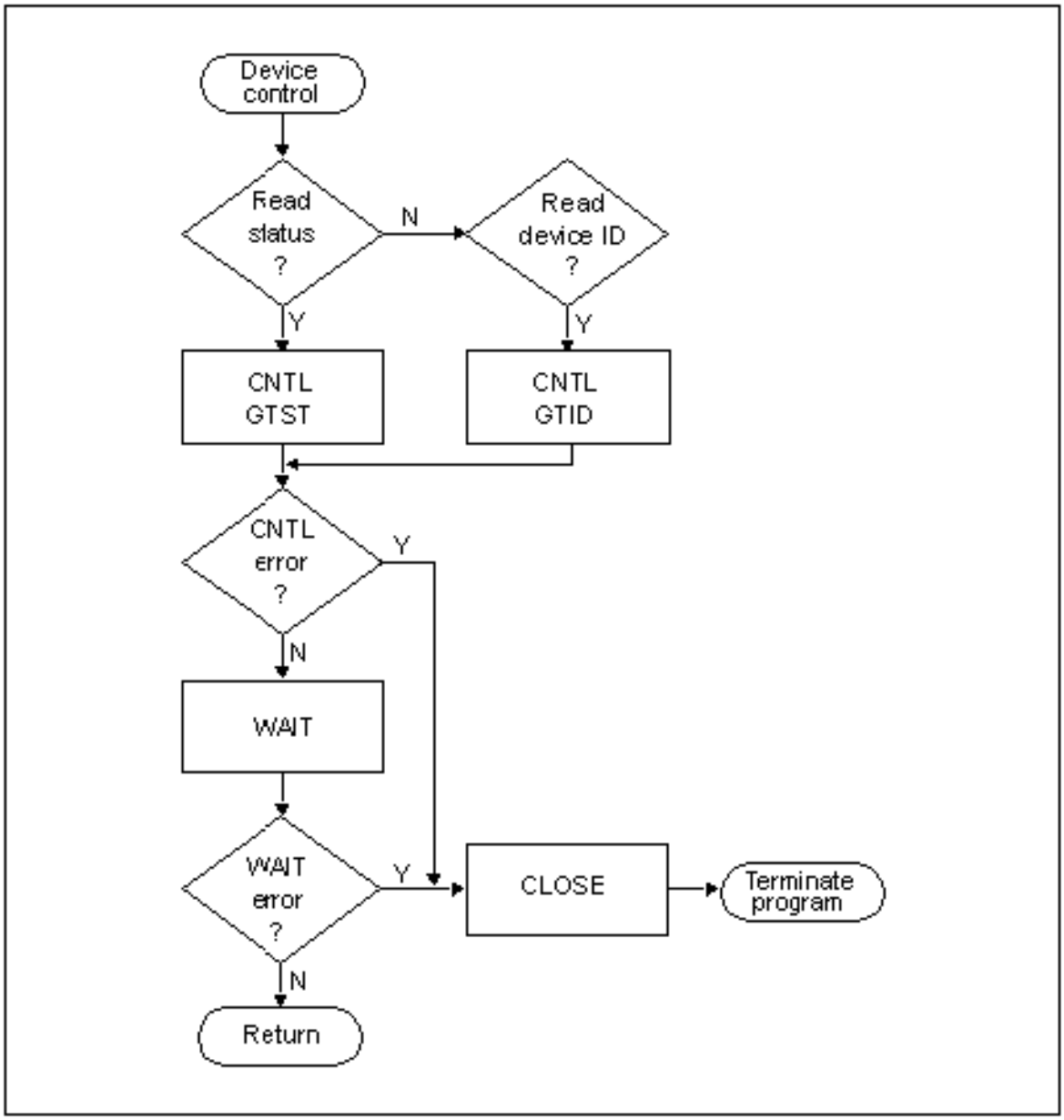
4.2.7.5 Output data



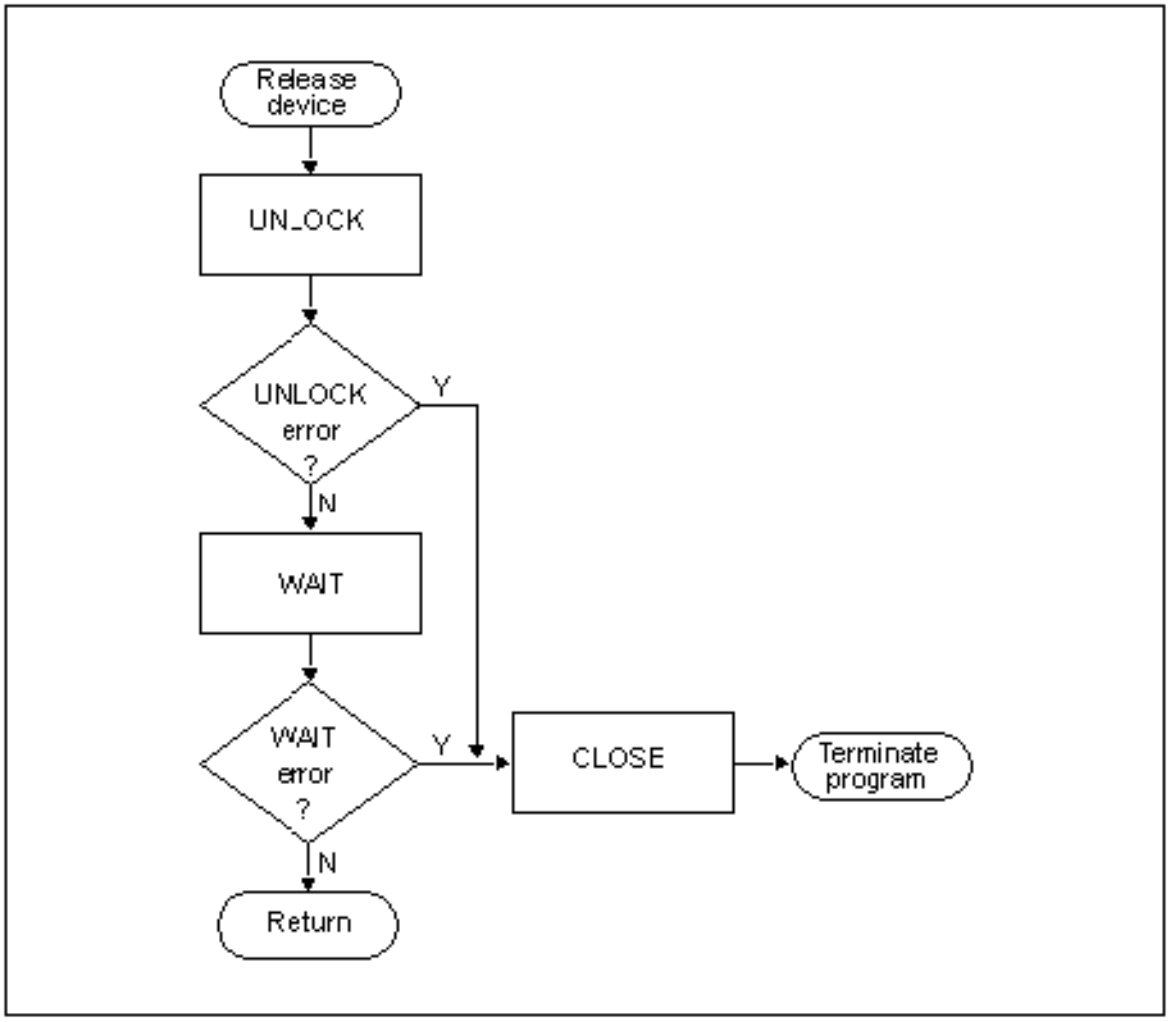
4.2.7.6 Read data



4.2.7.7 Device control



4.2.7.8 Release device



4.3 C programming interface

4.3.1 iocs_open – opening a printer

Synopsis

```
#include <xio.h>
int iocs_open (dev_name, fct_mod, oflag)
char *dev_name, *fct_mod;
int oflag;
```

Description

The OPEN command opens a printer for the required function module in the mode specified. A logical device number (> 0), which can be used to address the opened printer in subsequent commands, is returned to the application. If an error is detected, an error number (< 0) is returned.

dev_name

In this parameter the path name (*/dev...*) of the printer to be opened must be specified as a string.

fct_mod

In this parameter, the name of the desired function module is stored as a string. For printers controlled by default device control processes, *PRINTMOD* must be specified. For device control processes created by the user, the name of the function module can generally be freely chosen. It need only correspond to the entry in the *iocs.fmod* file (see the Section "Description of the iocs.fmod configuration

file").

oflag

In this parameter the routine synchronization must be specified.

X_NOSYNC The IOCS commands are processed asynchronously. Every IOCS command (with the exception of the CLOSE and CANCEL commands) must be fetched with a WAIT command.

X_SYNC The IOCS commands are processed synchronously, i.e. the WAIT command is issued implicitly. The return code is negative. Synchronous processing should be avoided where possible, since the WAIT command is generally interpreted as a "multiple wait".

Notes

If SAS printers are used in connection with DPTG terminals, the addressed printer is automatically locked for hardcopy commands executed at the workstation. This ensures that a print command is not disrupted by print commands executed at the workstation. If hardcopies are required, the printer must be released with a CLOSE command.

If the OPEN command is acknowledged without errors and processing is asynchronous, the next command must be a WAIT command. The WAIT command tells the application whether or not the printer could be opened correctly (*wtrev* = *XIO_OPEN*).

IF no error is reported in the WAIT command, the printer must be closed again before the application is ended. If the WAIT command reports an error produced by the OPEN command, the printer is closed again automatically.

With a PUT command, it is necessary to know the number of possible jobs that can be received before issuing a WAIT command when working with alternating buffers. This number is transferred to the external variable *xio_mseq*, which is defined in the *xio.h* file (see also the example on the following page).

Errors

The OPEN command cannot be executed if any of the following conditions applies:

EMFILE
The permitted number of open IOCS devices has been exceeded.

E_NOXIO
The IOCS has not been started.

E_NOENT
The required configuration is not available.

E_ISOPEN
The device has already been opened in this application.

E_IO1
General IOCS error.

E_COM
Error in communication between the processes involved in IOCS.

Example

```
#include <xio.h>
#include "xio_struct.h"
int bpt_open(dev_name, module, syncflag)
char *dev_name;                 /* path name of printer to be opened       */
char *module;                   /* function module */
```

```
int syncflag;                   /* command synchronization flag          */
{
extern int bpt_wait();
extern struct xio_wt order;
extern int maxsequence;      /* no. of possible PUT routines before WAIT */
int device_no;               /* logical device number                    */
int funct_val;
□□□□□device_no = iocs_open(dev_name, module, syncflag);
if (device_no < 0)
{
/* error in OPEN call */
}
else
{
order.wtdb = device_no;
if (syncflag == X_NOSYNC)
{
maxsequence = xio_mseq;
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* Evaluation of the structure xio_wt;       */
/* the structure xio_wt can only be evaluated */
/* if wtrev != 0 (see iocs_wait)             */
}
}
}
return (device_no);
}
```

4.3.2 iocs_close – closing a printer

Synopsis

```
#include <xio.h>
int iocs_close (p_db)
int p_db;
```

Description

The CLOSE command closes a previously with OPEN opened printer. The printer is released automatically and any pending commands are deleted. The value "0" is returned if the command is handled successfully. If an error is detected, an error number (< 0) is returned.

p_db
: The logical device number returned by *iocs_open* is transferred in this parameter.

Errors

The CLOSE command is not executed if one of the following conditions applies:

E_DEVNOP
: The device has not been opened.

E_XIOSTOP
: The IOCS has been stopped.

E_COM
Error in communication between the processes involved in IOCS.

Example

```
#include <xio.h>
#include "xio_struct.h"
int bpt_close()
{
extern struct xio_wt order;
int funct_val, device_no;
device_no = order.wtdb; /* wtdb contains log. dev. no. (see OPEN) */
funct_val = iocs_close(device_no);
if (funct_val < 0)
{
/* error in CLOSE call */
}
return(funct_val);
}
```

4.3.3 iocs_lock – locking printers

Synopsis

```
#include <xio.h>
int iocs_lock (nitems, p_dblist)
int nitems;
int (*p_dblist)[ ]; /* address of an integer array */
```

Description

An application can use a LOCK command to secure exclusive access to one or more printers. These printers are then locked and are unavailable to other applications. If the command is processed successfully, the value "0" is returned. If an error is detected, an error number (< 0) is returned.

nitems
The number of printers to be locked must be specified in this parameter.

p_dblist
This array contains a list of the devices numbers of printers to be locked.

Notes

Only one printer can be locked during synchronous routine processing (*nitems* = 1).

If the LOCK command is acknowledged without errors and processing is asynchronous, the next command must be a WAIT command. The WAIT command tells the application whether or not the printer(s) could be locked (*wtrev = XIO_LOCK*).

If a printer has already been locked by a LOCK command, the acknowledgment received from the WAIT command indicates an error (*wtinerr = E_NOLOCK*).

If a LOCK command is to lock a number of different printers, none of these printers can already be locked. If one of the printers in the list is already locked, none of the printers specified is locked by the command. The WAIT command can be used to determine which printers have already been locked (*wtouterr = E_ISLOCK*). A LOCK command can only be issued if all the other commands for this printer have been acknowledged with WAIT commands.

Errors

The LOCK command is not executed if any of the following conditions applies:

E_DEVNOP
The device has not been opened.

E_XIOSTOP

The IOCS has been stopped.

E_IO1
General IOCS error.

E_COM
Error in communication between the processes involved in IOCS.

E_NOLOCK
The device has not been locked.

E_FALSEF
The IOCS function call is invalid.

Example

```
#include <xio.h>
#include "xio_struct.h"
int bpt_lock(syncflag)
int syncflag;                   /* command synchronization flag */
{
extern int bpt_wait();
extern struct xio_wt order;
int funct_val, device_no;
device_no = order.wtdb;       /* logical device number (see OPEN) */
funct_val = iocs_lock(1, &device_no);
if (funct_val < 0)
{
/* error in LOCK call */
}
else
{
if (syncflag == X_NOSYNC)
{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* Evaluation of the structure xio_wt;        */
/* the structure xio_wt can only be evaluated */
/* if wtrev != 0 (see iocs_wait)             */
}
}
}
return(funct_val);
}
```

4.3.4 iocs_unlock – unlocking printers

Synopsis

```
#include <xio.h>
int iocs_unlock (nitems, p_dblist)
int nitems;
int (*p_dblist)[ ]; /* address of an integer array */
```

Description

An application can use the UNLOCK command to unlock several printers at once, regardless of whether they were originally locked individually or as a group. The value "0" is returned if the command is handled successfully. If an error is detected, an error number (< 0) is returned.

nitems
: The number of printers to be unlocked must be specified in this parameter.

p_dblist
: This array contains a list of the device numbers of the printers to be unlocked.

Notes

Only one printer can be specified during synchronous routine processing (*nitems* = 1).

If the UNLOCK command is acknowledged without errors and processing is asynchronous, the next routine must be a WAIT command. The WAIT command tells the application whether an attempt was made to unlock a printer that was not locked (*wtinerr* = *E_NOLOCK*). Any locked printers are unlocked.

CLOSE commands can be substituted for the necessary UNLOCK commands at the end of an application.

An UNLOCK command can only be issued if all commands for this printer have been acknowledged with WAIT commands.

Errors

The UNLOCK command is not executed if any of the following conditions applies:

E_DEVNOP
: The device has not been opened.

E_XIOSTOP
: The IOCS has been stopped.

E_IO1
: General IOCS error.

E_COM
: Error in communication between the processes involved in IOCS.

E_NOLOCK
: The device has not been locked.

E_FALSEF
: The IOCS function call is invalid.

Example

```
#include <xio.h>
#include "xio_struct.h"
int bpt_unlock(syncflag)
int syncflag;                   /* command synchronization flag */
{
extern int bpt_wait();
extern struct xio_wt order;
int funct_val, device_no;
device_no = order.wtdb;       /* logical device number (see OPEN) */
funct_val = iocs_unlock(1, &device_no);
if (funct_val < 0)
{
/* error in UNLOCK call */
}
else
{
if (syncflag == X_NOSYNC)
{
funct_val = bpt_wait();
```

```
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* Evaluation of the structure xio_wt;       */
/* the structure xio_wt can only be evaluated */
/* if wtrev != 0 (see iocs_wait)             */
}
}
}
return(funct_val);
}
```

4.3.5 iocs_put – outputting data ready for printing

Synopsis

```
#include <xio.h>
#include <xio_desc.h>
int iocs_put(p_db, ctlptr, dataptr)
int p_db;
struct xio_ctl *ctlptr;
char *dataptr;
```

Description

The PUT command can be used to perform different output functions:

- Output a data string (all printers)
- Set passbook parameters (ND90)
- Output ZEKOSA data (ND90)
- Eject passbook (ND90)
- Switch ZEKOSA formats (ND90)

During asynchronous processing, a routine sequence number is returned once a command has been processed successfully. This number is used to uniquely identify the routine for the WAIT command. If an error is detected, an error number (< 0) is returned.

In synchronous processing, the value "0" or *E_NEWST* is returned if the command has been processed successfully. *E_NEWST* indicates a change of status. If an error is detected, an error number (< 0) other than *E_NEWST* is returned. The status buffer can be read using the external pointer *xiopstat* (defined in *xio.h*).

p_db

The device number returned by *iocs_open* is transferred in this parameter.

ctlptr

A pointer to the control structure *xio_ctl* is transferred in this parameter. The format of the control structure is as follows:

```
struct xio_ctl
{
int bsize;
int nitems;
int fsize;
union xio_fct *pf;
};
```

bsize

This field is not used and should be initialized with "0".

nitems
: The number of characters to be output (< 65,000) is specified in this field.

fsize
: The various output functions and related parameters must be specified in function structures. All function structures (including those for the GET command – see Section "iocs_get reading data from a printer") are combined in the union *xio_fct*. The length of the function structure used must be specified in the field *fsize*.

pf
: A pointer to the union *xio_fct* must be defined in this field. Format of the union *xio_fct* (as far as is relevant to the PUT command):

```
union xio_fct
{
/* Outputting a data string                */
struct xio_prccc     pr_ccc;
/* Setting passbook parameters             */
struct xio_prbook    pr_book;
/* Outputting ZEKOSA data/ejecting a passbook */
struct xio_prputzek  pr_putzek;
/* Switching ZEKOSA formats                */
struct xio_prtypezek pr_typezek;
};
```

: The individual structures used for the various output functions are described on the following pages.

dataptr
: This parameter points to a buffer containing the data to be output.

Notes

If the PUT command is acknowledged without errors and processing is asyn-chronous, the next command must be a WAIT command (*wtrev = XIO_PUT*).

If the maximum command sequence number is greater than "0", the application can make use of alternate buffers to speed up printing. In this case, a WAIT command is needed once the maximum sequence number has been reached. The maximum sequence number+1 is defined in the file */usr/lib/iocs.fmod*.

Errors

The PUT command is not executed if any of the following conditions applies:

E_DEVNOP
: The device has not been opened.

E_XIOSTOP
: The IOCS has been stopped.

E_IO1
: General IOCS error.

E_COM
: Error in communication between the processes involved in IOCS.

E_FALSEF
: The IOCS function call is invalid.

E_MSEQ
: The maximum sequence number has been reached.

E_NEWST
: The device status has changed (in synchronous processing only).

4.3.5.1 Outputting a data string

This function is used to output user data ready for printing. The user data can contain printer control

characters prepared in accordance with a printer character class to be specified.

The function structure that outputs the data string, *xio_prccc*, is defined in the union *xio_fct* under the name *pr_ccc* and has the following format:

```
struct xio_prccc
{
int prfct;
int prclass;
};
```

prfct

The value *PR_CCC* must be specified in this field.

prclass

The control character class must be defined in this field:

The following settings must be made if characters that can be read by OCR software are to be printed on an ND90 (see also Section "Reading OCR data"):

Function	Setting
5 lines per inch Letter quality printing OCR-B	*LPI_5LQSWOB*(the line marker *7E* must be used with the font OCR- B)

When printing a passbook on the ND90, passbook format must be set beforehand (see Section "Setting passbook parameters") and a passbook/voucher must be inserted (e.g. using "Line finding").

Example of a function that outputs a data string to a printer:

```
#include <xio.h>
#include "xio_struct.h"
int bpt_putstr(char_no, syncflag, data)
int char_no;                 /* no. of characters to be output*/
int syncflag;                /* command synchronization flag  */
char *data;                  /* data to be output             */
{
extern int bpt_wait();
extern struct xio_wt order;
extern struct xio_ctl checkup;
extern union xio_fct function;
extern int lastsequence;       /* no. of last command issued */
extern int number;             /* number of open commands    */
extern int maxsequence;        /* no. of possible commands before WAIT */
int sequence_no;               /* command sequence number    */
int funct_val;
int device_no;
/* initialization of the control and function structures for outputting a data */
/* string */
checkup.bsize = 0;
checkup.nitems = char_no;
checkup.fsize = sizeof(function.pr_ccc);
checkup.pf = &function;
function.pr_ccc.prfct = PR_CCC;     /* function key */
function.pr_ccc.prclass = PR_EPSON; /* control character class */

device_no = order.wtdb; /* logical device number (see OPEN) */
sequence_no = iocs_put(device_no, &checkup, data);
```

```
if (sequence_no < 0)
{
/* error in PUT call */
}
else
{
if (syncflag == X_NOSYNC)
{
lastsequence = sequence_no;
number++;
/* WAIT call – issued when max. number is reached */
if (number == maxsequence)  /* maxsequence is set in OPEN */
{
do
{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* new number of open commands must be determined    */
/* because, depending on the maximum sequence number */
/* in a WAIT, several commands can be acknowledged   */
if (order.wtinps <= lastsequence)
{
number = lastsequence - order.wtinps;
}
else
{
number = lastsequence - order.wtinps + maxsequence;
}
if (order.wtst != 0) /* marker for status changes */
{
/* CNTL call for status query */
}
}
□□□□□□□□□□□□□□□□□□□}□while (order.wtrev != XIO_PUT);
}
}
}
return (sequence_no);
}
```

4.3.5.2 Setting passbook parameters

This function is used to set passbook parameters with the aid of a function structure. Alternatively, settings can be made in the data string using the ESC sequence *SPBM* (see the Section "Special control sequences for the ND90").

The structure used to set passbook parameters, *xio_prbook*, is defined in the union *xio_fct* under the name *pr_book* and has the following format:

```
struct xio_prbook
{
int prfct;
```

```
int prfpar1;
...
int prfpar14;
};
```

prfct

The value *PR_BOOKPAR* must be specified in this field.

prfpar

The following parameters must be specified in the fields *prfpar1* through *prfpar14*:

prfpar1 Position of the line marker in characters relative to the left-hand margin at a character pitch of 10 cpi (0 <= *prfpar1* <= 90).

prfpar2 Vertical position of the first booking line relative to the passbook header at a line density of 1/60th of an inch (13 <= *prfpar2* <= 3333).

prfpar3 Vertical position of the footer relative to the passbook header at a line density of 1/60th of an inch (*prfpar2* < *prfpar3* <= 3333).

prfpar4 Number of lines to be searched through (1 <= *prfpar4* <= 99).

prfpar5 First line of a non-printable area relative to the first booking line (0 <= *prfpar5* < *prfpar4*).

prfpar6 Number of locked lines (0 <= *prfpar6* < *prfpar4*).

prfpar7 At present, this field is not evaluated. an inch).

prfpar8 Start position of the first read block in the footer at 10 cpi (0 <= *prfpar8* <= 90).

prfpar9 Stop position of the first read block in the footer at 10 cpi (*prfpar8* < *prfpar9* <= 90).

prfpar10 Start position of the second read block of the booking line at 10 cpi (0 <= *prfpar10* <= 90).

prfpar11 Stop position of the second read block of the booking line at 10 cpi (*prfpar10* < *prfpar11* <= 90).

prfpar12 Start position of the first read block of the booking line at 10 cpi (0 <= *prfpar12* <= 90).

prfpar13 Stop position of the first read block of the booking line at 10 cpi (*prfpar12* < *prfpar13* < *prfpar10*).

prfpar14 A "1".

Example of a function that sets passbook parameters:

```
#include <xio.h>
#include "xio_struct.h"
int bpt_putsbp(syncflag)
int syncflag;                   /* command synchronization flag */
{
extern int bpt_wait();
extern struct xio_wt order;
extern xio_ctl checkup;
```

```
extern union xio_fct function;
extern int lastsequence;       /* no. of the last command issued */
extern int number;             /* no. of open commands           */
extern int maxsequence;        /* no. of possible commands before WAIT */
int sequence_no;               /* command sequence number        */
int funct_val, device_no;
/* initialization of control / function structures for setting passbook parameters */
checkup.bsize = 0;
checkup.nitems = 0;
checkup.fsize = sizeof(function.pr_book);
checkup.pf = &function;
function.pr_book.prfct = PR_BOOKPAR;
function.pr_book.prfpar1   = /* function parameter 1 */ ;
.
.
.
function.pr_book.prfpar14  = /* function parameter 14 */ ;
device_no = order.wtdb; /* logical device no. (see OPEN) */
sequence_no = iocs_put(device_no, &checkup, NULL);
if (sequence_no < 0)
{
/* error in PUT call */
}
else
□
□□□□□{
if (syncflag == X_NOSYNC)
{
lastsequence = sequence_no;
number++;
/* WAIT call – issued when max. number is reached */
if (number == maxsequence)  /* maxsequence is set in OPEN */
{
do
{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* new number of open commands must be determined    */
/* because, depending on the maximum sequence number */
/* in a WAIT, several commands can be acknowledged   */
if (order.wtinps <= lastsequence)
{
number = lastsequence - order.wtinps;
}
else
{
number = lastsequence - order.wtinps + maxsequence;
}
if (order.wtst != 0) /* marker for status changes */
```

```
{
/* CNTL call for status query */
}
}
□□□□□□□□□□□□□□□□□□□}□while (order.wtrev != XIO_PUT);
}
}
}
return (sequence_no);
}
```

4.3.5.3 Outputting ZEKOSA data

This function is used to write data to the magnetic strip in a passbook.

The structure *xio_prputzek*, for outputting ZEKOSA data, is defined under the name *pr_putzek* in the union *xio_fct* and has the following format:

```
struct xio_prputzek
{
int zefct;
};
```

zefct

The value *ZEK_PUT* must be specified in this field.

The data in the data string must be written in ASCII (ISO 8-bit). The standards quoted support the following characters:

Standard	Characters
DIN□	*0 1 2 3 4 5 6 7 8 9 : < > ; ?□*
IBM	*0 1 2 3 4 5 6 7 8 9 : < > = ?*

The character "?" serves as the terminating character in both standards.

Example of a function that outputs ZEKOSA data:

```
□#include <xio.h>
#include "xio_struct.h"
int bpt_putzek(char_no, syncflag, data)
int char_no;                    /* no. of characters to be output*/
int syncflag;                   /* command synchronization field */
char *data;                     /* print data to be output       */
{
extern int bpt_wait();
extern struct xio_wt order;
extern struct xio_ctl checkup;
extern union xio_fct function;
extern int lastsequence;        /* no. of the last command issued */
extern int number;              /* no. of open commands           */
extern int maxsequence;         /* no. of possible commands before WAIT */
int sequence_no;                /* command sequence number        */
int funct_val, device_no;
/* initialization of the control and function structures for outputting ZEKOSA data */
checkup.bsize = 0;
checkup.nitems = char_no;
checkup.fsize = sizeof(function.pr_putzek);
checkup.pf = &function;
function.pr_putzek.zefct = ZEK_PUT;
```

```
device_no = order.wtdb; /* logical device number (see OPEN) */
sequence_no = iocs_put(device_no, &checkup, data);
if (sequence_no < 0)
{
/* error in PUT call */
}
else
{
if (syncflag == X_NOSYNC)
{
lastsequence = sequence_no;
number++;
/* WAIT call – issued when max. number is reached */
if (number == maxsequence)  /* maxsequence is set in OPEN */
{
do
□
□□□□□□□□□□□□□□□□□{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* new number of open commands must be determined    */
/* because, depending on the maximum sequence number */
/* in a WAIT, several commands can be acknowledged   */
if (order.wtinps <= lastsequence)
{
number = lastsequence - order.wtinps;
}
else
{
number = lastsequence - order.wtinps + maxsequence;
}
if (order.wtst != 0) /* marker for status changes */
{
/* CNTL call for status query */
}
}
□□□□□□□□□□□□□□□□□}□while (order.wtrev != XIO_PUT);
}
}
}
return (sequence_no);
}
```

4.3.5.4 Ejecting a passbook

This function ejects a passbook that has been inserted.

The structure *xio_prputzek*, for ejecting the passbook, is defined under the name *pr_putzek* in the union *xio_fct* and has the following format:

```
struct xio_prputzek
{
```

```
int zefct;
};
```

zefct

The value *ZEK_PBREL* must be specified in this field.

Example of a function that ejects a passbook:

```
#include <xio.h>
#include "xio_struct.h"
int bpt_putasb(syncflag)
int syncflag;                    /* routine synchronization flag    */
{
extern int bpt_wait();
extern struct xio_wt order;
extern struct xio_ctl checkup;
extern union xio_fct function;
extern int lastsequence;        /* no. of the last routine issued  */
extern int number;              /* no. of open routines            */
extern int maxsequence;         /* no. of possible routines before WAIT */
int sequence_no;                /* routine sequence number         */
int funct_val, device_no;
/* initialization of the control and function structures for ejecting the passbook */
checkup.bsize = 0;
checkup.nitems = 0;
checkup.fsize = sizeof(function.pr_putzek);
checkup.pf = &function;
function.pr_putzek.zefct = ZEK_PBREL;
device_no = order.wtdb; /* logical device number (see OPEN) */
sequence_no = iocs_put(device_no, &checkup, NULL);
if (sequence_no < 0)
{
/* error in PUT call */
}
else
{
if (syncflag == X_NOSYNC)
{
lastsequence = sequence_no;
number++;
/* WAIT call – issued when max. number is reached */
if (number == maxsequence)  /* maxsequence is set in OPEN */
{
do
{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* new number of open commands must be determined    */
/* because, depending on the maximum sequence number */
/* in a WAIT, several commands can be acknowledged   */
if (order.wtinps <= lastsequence)
```

```
{
number = lastsequence - order.wtinps;
}
□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□else
{
number = lastsequence - order.wtinps + maxsequence;
}
if (order.wtst != 0) /* marker for status changes */
{
/* CNTL call for status query */
}
}
□□□□□□□□□□□□□□□□□□}□while (order.wtrev != XIO_PUT);
}
}
}
return (sequence_no);
}
```

4.3.5.5 Switching ZEKOSA formats

This function is used to define the ZEKOSA format.

The structure *xio_prtypezek*, for switching the ZEKOSA format, is defined under the name *pr_typezek* in the union *xio_fct* and hasthe following format:

```
struct xio_prtypezek
{
int zefct;
int zetype;
};
```

zefct

The value *ZEK_ST* must be specified in this field.

zetype

One of the following formats must be specified in this field:

DIN□	DIN format, track length 45 characters (default)□
IBM	IBM format, track length 36 characters

Example of a function that switches the ZEKOSA format:

```
□#include <xio.h>
#include "xio_struct.h"
int bpt_putuzf(char_no, syncflag, data)
int syncflag;                   /* routine synchronization flag    */
{
extern int bpt_wait();
extern struct xio_wt order;
extern struct xio_ctl checkup;
extern union xio_fct function;
extern int lastsequence;        /* no. of the last routine issued  */
extern int number;              /* no. of open routines            */
extern int maxsequence;         /* no. of possible routines before WAIT */
int sequence_no;                /* routine sequence number         */
int funct_val, device_no;
□□□/*□□  initialization of the control and function structures for switching□□□□*/
```

```
/*  ZEKOSA formats
*/
checkup.bsize = 0;
checkup.nitems = 0;
checkup.fsize = sizeof(function.pr_typezek);
checkup.pf = &function;
function.pr_typezek.zefct = ZEK_ST;
function.pr_typezek.zetype = DIN;
device_no = order.wtdb; /* logical device no. (see OPEN) */
sequence_no = iocs_put(device_no, &checkup, NULL);
if (sequence_no < 0)
{
/* error in PUT call */
}
else
{
if (syncflag == X_NOSYNC)
{
lastsequence = sequence_no;
number++;
/* WAIT call – issued when max. number is reached */
if (number == maxsequence)  /* maxsequence is set in OPEN */
{
do
{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* new number of open commands must be determined    */
/* because, depending on the maximum sequence number */
/* in a WAIT, several commands can be acknowledged   */
if (order.wtinps <= lastsequence)

{
number = lastsequence - order.wtinps;
}
else
{
number = lastsequence - order.wtinps + maxsequence;
}
if (order.wtst != 0) /* marker for status changes */
{
/* CNTL call for status query */
}
}
} while (order.wtrev != XIO_PUT);
}
}
}
return (sequence_no);
```

}

4.3.6 iocs_get – reading data from a printer

Synopsis

```
#include <xio.h>
#include <xio_desc.h>
int iocs_get (p_db, ctlptr, dataptr)
int p_db;
struct xio_ctl *ctlptr;
char *dataptr;
```

Description

The GET routine is used to read data from printers. The syntax used in the GET routine is formally the same as that in a PUT routine and is designed to handle the following read functions:

- Line finding (ND90)
- Read booking line (ND90)
- Read footer (ND90)
- Read OCR data (ND90)
- Read ZEKOSA data (ND90)

During asynchronous processing, a routine sequence number is returned when a routine has been processed successfully. This number is used to uniquely identify the routine for the WAIT. If an error is detected, an error number (< 0) is returned.

During synchronous processing, the value "0" or *E_NEWST* is returned when a routine has been processed successfully. *E_NEWST* indicates a change of status. If an error is detected, an error number (< 0) other than *E_NEWST* is returned. The status buffer can be read using the external pointer *xiopstat* (defined in *xio.h*).

p_db

The device number returned by *iocs_open* is transferred in this parameter.

ctlptr

A pointer to the control structure *xio_ctl* is transferred in this parameter. The format of the control structure is as follows:

```
struct xio_ctl
{
int bsize;
int nitems;
int fsize;
union xio_fct *pf;
};
```

bsize — The length of the data buffer (< 2000) indicated by *dataptr* is transferred in this field.

nitems — After a WAIT command, the number of characters read is returned in this field.

fsize — The various read functions and their parameters must be specified in function-specific structures. All function structures (including those for the PUT command – see Section "iocs_put outputting data ready for printing") are combined in the union *xio_fct*. The length of the structure used must be defined in the field *fsize*.

pf — A pointer to the union *xio_fct* must be specified in this field. The format of the union is as follows (as far as is relevant

for the GET command):

```
union xio_fct□
{□
/* Line finding                */□
struct xio_prlfin   pr_lfin;□
/* Reading a booking line/footer */□
struct xio_prrdbl   pr_rdbl;□
/* Reading OCR data            */□
struct xio_prgtocr  pr_gtocr;□
/* Reading ZEKOSA data         */□
struct xio_prgetzek pr_getzek;□
};
```

The individual structures for the various read functions are described on the following pages.

datapr
: This parameter indicates a buffer to which the data that has been read is transferred after a WAIT command. The buffer is returned in structured form with the functions "Line finding", "Read booking line" and "Read footer".

Notes

If the GET command is acknowledged without errors, and processing is asynchronous, the next command must be a WAIT command (*wtrev*□ = *XIO_GET*). In contrast to the PUT command, the GET command does not support the use of alternate buffers.

Errors

The GET command is not executed if any one of the following conditions applies:

E_DEVNOP
: The device has not been opened.

E_XIOSTOP
: The IOCS has been stopped.

E_IO1
: General IOCS error.

E_COM
: Error in communication between the processes involved in IOCS.

E_FALSEF
: The IOCS function call is invalid.

E_MSEQ
: The maximum sequence number has been reached.

E_NEWST
: The device status has changed (in synchronous processing only).

4.3.6.1 Line finding

This function is used to feed in a voucher or passbook and read the contents of the passbook footer and the last booking line. The print head is set to the physical starting position for printing on the first free line in the voucher or passbook. The passbook format must be set before this routine is used (see Section "Setting passbook parameters").

The function structure *xio_prlfin*, for line finding, is defined under the name pr_lfin in the union *xio_fct* and has the following format:

```
struct xio_prlfin
{
int prfct;
};
```

prfct
: The value *PR_LFIN* must be specified in this field.

After the WAIT command, the data read is transferred in the structure *prbl_data* to which *dataptr* points. The structure has the following format:

```
#define X_PRBLEN 160
struct prbl_data
{
unsigned short prline;
unsigned short prlpbl1;
unsigned short prlpbl2;
unsigned short prlpfl;
char prbuffer[X_PRBLEN];
};
```

prline
: The line number of the last booking line is transferred in this field.

prlpbl1
: The length of the data in the first read block of the last booking line is transferred in this field. The data starts at the position *prbuffer[0]*.

prlpbl2
: The length of the data in the second read block of the last booking line is transferred in this field. The data starts at the position *prbuffer[prlpbl1]*.

prlpfl
: The length of the data in the passbook footer is transferred in this field. The data starts at the position *prbuffer[prlpbl1+prlpbl2]*.

prbuffer
: The data read is stored in this array.

Example of a function that carries out line finding:

```
#include <xio.h>
#include "xio_struct.h"
int bpt_getlf(syncflag)
int syncflag;                   /* routine synchronization flag */
{
extern int bpt_wait();
extern struct xio_wt order;
extern struct prbl_data line;
struct prbl_data *line_ptr;
extern struct xio_ctl checkup;
extern union xio_fct function;
int funct_val, device_no;
/* initialization of the control and function structures for line finding */
line_ptr = &line;
checkup.bsize = sizeof(line);
checkup.fsize = sizeof(function.pr_lfin);
checkup.pf = &function;
function.pr_lfin.prfct = PR_LFIN;
device_no = order.wtdb;     /* logical device no. (see OPEN)    */
funct_val = iocs_get(device_no, &checkup, (char *)line_ptr);
if (funct_val < 0)
{
/* error in GET call */
}
```

```
else
{
if (syncflag == X_NOSYNC)
{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* Evaluation of the structure xio_wt; the structure xio_wt */
/* can only be evaluated if wtrev != 0 (see iocs_wait)      */
}
}
}
return (funct_val);
}
```

4.3.6.2 Reading a booking line/footer

This function is used to read the contents of the current booking line or the footer. The passbook format must be set before this routine is used (see Section "Setting passbook parameters") and a passbook has to have been fed in.

The structure *xio_prrdbl*, for reading booking lines/footers, is defined under the name *pr_rdbl* in the union *xio_fct* and has the following format:

```
struct xio_prrdbl
{
int prfct;
};
```

prfct

For the "Reading a booking line" function the value *PR_RDBL* must be specified in this field. For the "Reading a footer" function the value *PR_RDFL* must be specified in this field.

After the WAIT command, the data read is transferred in the structure *prbl_data* to which *dataptr* points. The structure has the following format:

```
#define X_PRBLEN 160
struct prbl_data
{
□□unsigned short prline;  /* irrelevant for both functions */
□□unsigned short prlpbl1; /* irrelevant for 'read footer' □*/
□□unsigned short prlpbl2; /* irrelevant for 'read footer' □*/
□□unsigned short prlpfl; /* irrelevant for 'read bookingline' */
□□char prbuffer[X_PRBLEN];
};
```

prlpbl1

The length of the data in the first read block of the booking line is transferred in this field. The data starts at the position *prbuffer[0]*.

prlpbl2

The length of the data in the second read block of the booking line is transferred in this field. The data starts at the position *prbuffer[prlpbl1]*.

prlpfl

The length of the data in the footer is transferred in this field.

prbuffer

The data read is stored in this array.

The contents of the fields marked as "irrelevant" are set to zero for the functions described.

Example of a function that reads a booking line:

```
#include <xio.h>
#include "xio_struct.h"
int bpt_getbzl(syncflag)
int syncflag;                    /* routine synchronization flag   */
{
extern int bpt_wait();
extern struct xio_wt order;
extern struct prbl_data line;
struct prbl_data *line_ptr;
extern struct xio_ctl checkup;
extern union xio_fct function;
int funct_val, device_no;
/* initialization of the control and function structures for reading a booking line */
line_ptr = &line;
checkup.bsize = sizeof(line);
checkup.fsize = sizeof(function.pr_rdbl);
checkup.pf = &function;
function.pr_rdbl.prfct = PR_RDBL; /* specify PR_RDFL to read footer */
device_no = order.wtdb;     /* logical device no. (see OPEN)    */
funct_val = iocs_get(device_no, &checkup, (char *)line_ptr);
if (funct_val < 0)
{
/* error in GET call */
}
else
{
if (syncflag == X_NOSYNC)
{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* Evaluation of the structure xio_wt;        */
/* the structure xio_wt can only be evaluated */
/* if wtrev != 0 (see iocs_wait)             */
}
}
}
return (funct_val);
}
```

Example of an output procedure for a structure line of type *prbl_data*:

```
#include <xio.h>
#include "xio_struct.h"
void prbl_output()
{
extern struct prbl_data line;
```

```
printf("line number□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□: %hd\n", line.prline);
printf("length of 1st read block of posting line□□□: %hd\n", line.prlpbl1);
printf("length of 2nd read block of posting line□□□: %hd\n", line.prlpbl2);
printf("length of data in footer□□□□□□□□□□□□□□□□□□□□□: %hd\n", line.prlpfl);
printf("contents of 1st block□□□□□□□□□□□□□□□□□□□□□□□□□: %*s\n",
line.prlpbl1, line.prbuffer[0]];
printf("contents of 2nd block□□□□□□□□□□□□□□□□□□□□□□□□□: %*s\n",
line.prlpbl2, line.prbuffer[line.prlpbl1]);
printf("contents of footer□□□□□□□□□□□□□□□□□□□□□□□□□□□□: %*s\n",
line.prlpfl, line.prbuffer[line.prlpbl1 + line.prlpbl2]);
}
```

4.3.6.3 Reading OCR data

This function allows data to be read by printers with an OCR reading facility. The data at the current vertical position is read. Before executing the routine, you must switch to coding line processing by issuing the ESC sequence *SCLP* in a PUT command (see the Section "Special control sequences for the ND90").

The function structure *xio_prgtocr*, for reading OCR data, is defined under the name *pr_gtocr* in the union *xio_fct* and has the following format:

```
struct xio_prgtocr
{
int prfct;
int prpos;
int prnitems;
};
```

prfct
: The value *PR_GTOCR* must be specified in this field.

prpos
: The absolute start position for the read must be specified in this field in characters of 1/10 of an inch.

prnitems
: The number of characters to be read (max. 100) must be specified in this field.

Example of a function that reads OCR data:

```
□#include <xio.h>
#include "xio_struct.h"
int bpt_getocr(syncflag, read_pos, characters)
int syncflag;            /* routine synchronization flag      */
int read_pos, characters; /* start pos. for read, no. of chars.*/
{
int extern bpt_wait();
extern struct xio_wt order;
extern struct xio_ctl checkup;
extern union xio_fct function;
char buffer[X_PRBLEN];     /* buffer for data to be read */
int funct_val, device_no;
/* initialization of the control and function structures for reading OCR data */
checkup.bsize = sizeof(buffer);
checkup.fsize = sizeof(function.pr_gtocr);
checkup.pf = &function;
function.pr_gtocr.prfct = PR_GTOCR;
function.pr_gtocr.prpos = read_pos;
function.pr_gtocr.prnitems = characters;
device_no = order.wtdb;
```

```
funct_val = iocs_get(device_no, &checkup, &buffer);
if (funct_val < 0)
{
/* error in GET call */
}
else
{
if (syncflag == X_NOSYNC)
{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* Evaluation of the structure xio_wt;       */
/* the structure xio_wt can only be evaluated */
/* if wtrev != 0 (see iocs_wait)             */
}
}
}
return (funct_val);
}
```

4.3.6.4 Reading ZEKOSA data

This function is used to read the data on the magnetic strip of a passbook.

The function structure *xio_prgetzek*, for reading ZEKOSA data, is defined under the name *pr_getzek* in the union *xio_fct* and has the following format:

```
struct xio_prgetzek
{
int zefct;
int zeclass;
};
```

zefct

The value *ZEK_GET* must be specified in this field.

zeclass

One of the following control character classes must be specified in this field:

ZEK_ZEKOSA	The data read is transferred to the application without start/stop characters and LCRs (length redundant characters).
ZEK_NATIVE	The data read is transferred to the application without analysis, i.e. with start/stop and LCR characters.

Example of a function that reads ZEKOSA data:

```
#include <xio.h>
#include "xio_struct.h"
int bpt_getzek(syncflag, read_pos, characters)
int syncflag;          /* routine synchronization flag     */
int read_pos, characters; /* start pos. for read, no. of chars.*/
{
```

```
extern int bpt_wait();
extern struct xio_wt order;
extern struct xio_ctl checkup;
extern union xio_fct function;
char buffer[160];     /* buffer for the data read */
int funct_val, device_no;
/* initialization of the control and function structures for reading ZEKOSA data */
checkup.bsize = sizeof(buffer);
checkup.fsize = sizeof(function.pr_getzek);
checkup.pf = &function;
function.pr_getzek.zefct = ZEK_GET;
function.pr_getzek.zeclass = ZEK_ZEKOSA;
device_no = order.wtdb;
funct_val = iocs_get(device_no, &checkup, &buffer);
if (funct_val < 0)
{
/* error in GET call */
}
else
{
if (syncflag == X_NOSYNC)
{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* Evaluation of the structure xio_wt;       */
/* the structure xio_wt can only be evaluated */
/* if wtrev != 0 (see iocs_wait)             */
}
}
}
return (funct_val);
}
```

4.3.7 iocs_cntl – control function for printers

Synopsis

```
#include <xio.h>
int iocs_cntl (p_db, command, arg)
int p_db;
int command;
struct xio_arg *arg;
```

Description

The CNTL command can be used to read the parameters of an open printer and to cancel PUT routines. If the command has been processed successfully, the value "0" is returned. If an error is detected, an error number (< 0) is returned.

p_db

The device number returned by *iocs_open* is transferred in this parameter.

command

The required routine is specified in this parameter:

X_GTID	Reads the device ID
X_GTST	Reads the device status
X_OUTPCAN	Cancels PUT routines
X_CHECK	Monitors PUT routines

arg

A pointer to the argument structure *xio_arg* must be set in this parameter.The format of the argument structure is as follows:

```
struct xio_arg  /* not interpreted with X_OUTPCAN */
{
  int datl;     /* not interpreted with X_CHECK   */
  int nitems;
  char *dat_buf; /* not interpreted with X_CHECK   */
};
```

datl
: The length of the data buffer indicated by *arg* is specified in this field. This length is usually set using the C operator *sizeof*.

nitems
: The number of characters read is returned after a WAIT command in this field. With an *X_CHECK*, the field is used to specify the waiting period.

dat_buf
: A pointer to a data structure must be set in this field. The device ID or the status read is transferred to this data structure after a WAIT command.

Notes

Synchronization is required before the CNTL command is executed during asynchronous processing (except *X_OUTPCAN*). If the CNTL command is acknowledged without errors, the next routine must be a WAIT command. An *X_OUTPCAN* does not require a WAIT command, but the routine to be canceled must be fetched with a WAIT command.

Errors

The CNTL command is not executed if any of the following conditions applies:

E_DEVNOP
: The device has not been opened.

E_XIOSTOP
: The IOCS has been stopped.

E_IO1
: General IOCS error.

E_COM
: Error in communication between the processes involved in IOCS.

E_FALSEF
: The IOCS function call is invalid.

4.3.7.1 Reading the device ID

This function is used to read the printer ID, ZEKOSA ID and OCR ID of a printer. ZEKOSA and OCR IDs are only supported by the ND90.

Reading the printer ID of any printer

If the printer ID of a printer of any type is to be read, a pointer to the data structure *xio_prid* must be set in the

field *dat_buf* of the argument structure *xio_arg*. The format of the data structure is as follows:

```
struct xio_prid
{
□□unsigned short prdev; □□□/* device type □□□□□□□□□□□□□□□□□□□□□□*/
□□unsigned short prfree1; □/* free □□□□□□□□□□□□□□□□□□□□□□□□□□□□*/
□□unsigned short prtype; □□/* printer type□□□□□□□□□□□□□□□□□□□□□*/
□□unsigned short prfree2; □/* free□□□□□□□□□□□□□□□□□□□□□□□□□□□ □*/
□□unsigned short prsize_a; /* printing width, journal □□□□□□□□*/
□□unsigned short prsize_b;□/* printing width, print station 1□*/
□□unsigned short prsize_c;□/* printing width, print station 2□*/
□□unsigned short prsize_d;□/* printing width, voucher □□□□□□□□*/
};
```

prdev
: The value *X_PRINTER* is transferred in this field.

prtype
: The value defined in the file *xio.h* for the type of printer is transferred in this field.

prsize_*x*
: The printing widths of the individual print stations for output at 10 cpi are transferred in the fields *prsize_a*, *prsize_b*, *prsize_c* and *prsize_d*.

Reading the ZEKOSA and OCR IDs of an ND90

In addition to the usual printer ID, the ND90 has a ZEKOSA ID and an OCR ID. Two other data structures are therefore provided in addition to the data structure *xio_prid*. The three structures for reading the ND90's device IDs are defined in the superordinate structure *xio_idnd90*. A pointer to this structure must thus be set in the field *dat_buf* of the argument structure *xio_arg*. The format of this superordinate structure is as follows:

```
struct xio_idnd90
{
struct xio_prid pr_id;  /* read printer ID (see above) */
struct xio_zeid ze_id;  /* read ZEKOSA ID              */
struct xio_drid dr_id;  /* read OCR ID                 */
};
```

Format of the data structure *xio_zeid*:

```
struct xio_zeid
{
unsigned short zedev;    /* device type  */
unsigned short zefree1;  /* free         */
unsigned short zetype;   /* printer type */
unsigned short zefree2;  /* free         */
unsigned short zesize1;  /* track length */
unsigned short zefree3;  /* free         */
unsigned short zefree4;  /* free         */
unsigned short zefree5;  /* free         */
};
```

zedev
: The value for *X_ZEKOSA* is transferred in this field.

zetype
: The value for the printer type *ZEK_DIN* or *ZEK_IBM* is transferred in this field.

zesize1
: The appropriate track length (45/36) for the format is transferred in this field.

Format of the data structure *xio_drid*:

```
struct xio_drid
```

```
{
unsigned short drdev;    /* device type  */
unsigned short drfree1;  /* free         */
unsigned short drtype;   /* printer type */
unsigned short drfree2;  /* free         */
unsigned short drsize1;  /* free         */
unsigned short drfree3;  /* free         */
unsigned short drfree4;  /* free         */
unsigned short drfree5;  /* free         */
};
```

drdev

The value for *X_DOCREAD* is transferred in this field.

drtype

The value for *DRD_ND90* is transferred in this field.

If only the printer ID of the ND90 need be read, the length of the printer structure can be specified in the field *datl* of the argument structure *xio_arg*, e.g. sizeof(struct xio_prid).

If the ZEKOSA and/or OCR ID of the ND90 must also be read, the maximum data buffer length must be specified in the *field dat1* of the argument structure *xio_arg*, e.g. sizeof(struct xio_stnd90).

Example of a function that reads the device ID:

```
#include <xio.h>
#include "xio_struct.h"
int bpt_ident(syncflag)
int syncflag;          /* routine synchronization flag */
{
extern int bpt_wait();
extern struct xio_wt order;
extern struct xio_idnd90 nd90ident;
struct xio_idnd90 *nd90_ptr;
extern struct xio_arg argument;
int funct_val, device_no;
int command = X_GTID;
/* initialization of the argument structure for reading the device ID */
nd90_ptr = &nd90ident;
argument.datl = sizeof(nd90ident);
argument.nitems = 0;
argument.dat_buf = (char *)nd90_ptr;
    device_no = order.wtdb;  /* logical device no. (see OPEN) */
funct_val = iocs_cntl(device_no, command, &argument);
if (funct_val < 0)
    {
/* error in CNTL call */
}
else
{
if (syncflag == X_NOSYNC)
{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
```

```
{
/* Evaluation of the structure xio_wt;       */
/* the structure xio_wt can only be evaluated */
/* if wtrev != 0 (see iocs_wait)             */
}
}
}
return (funct_val);
}
```

4.3.7.2 Reading the device status

This function can be used to read the printer status, ZEKOSA status and OCR status of a printer. The ZEKOSA and OCR statuses are only supported by the ND90.

Reading the status of a printer of any type

If the status of a printer is to be read, a pointer to *xio_prstat* must be set in the field *dat_buf* of the argument structure *xio_arg*. The format of the data structure is:

```
struct xio_prstat
{
unsigned char pronl;  /* online/offline  */
unsigned char prst1;  /* paper/ribbon    */
unsigned char prst2;  /* form            */
};
```

pronl

One of the following values is reported in this field:

S_ONL	Device is online
S_OFFL	Device is offline

prst1

One of the following values is reported in this field:

S_PEND	Voucher not present
S_PIN	Voucher in position
S_CEND	Ribbon out

prst2

One of the following values is reported in this field:

S_EHOR	Form exceeded horizontally
S_EVER	Form exceeded vertically

Reading the ZEKOSA and OCR status of an ND90

In addition to the usual printer status, the ND90 has a ZEKOSA status and an OCR status that can be queried. Two other data structures are therefore provided in addition to the data structure *xio_prstat*. The three structures for querying the ND90's device status are defined in the superordinate structure *xio_stnd90*. A pointer to this structure must thus be set in the field *dat_buf* of the argument structure *xio_arg*. The format of the superordinate data structure is as follows:

```
struct xio_stnd90
{
    struct xio_prstat pr_stat; /* read printer status          */
                               /* (see above)                  */
    struct xio_zestat ze_stat; /*  read ZEKOSA status          */
```

□□□□struct xio_drstat dr_stat; /* read OCR status□□□□□□□□□□□□□□□□*/
};

Format of the data structure *xio_zestat*:

```
struct xio_zestat
{
unsigned char zeonl;   /* online/offline    */
unsigned char zepbin;  /* voucher status    */
unsigned char unused;  /* free              */
unsigned char zekosa;  /* extended status   */
unsigned char free1;   /* free              */
unsigned char free2;   /* free              */
unsigned char free3;   /* free              */
};
```

zeonl

One of the following values is reported in this field:

S_ONL□ Device is online□
S_OFFL Device is offline

zepbin

One of the following values is reported in this field:

S_NOPB□ No passbook in position□
S_PBIN□ Passbook in position□
S_OLDPB Passbook not removed

zekosa

Internal device errors are reported in this field (if necessary, inform Siemens Nixdorf Service):

Command	**No.**	**Meaning**
Output ZEKOSA data	30	no error
	>30	write error (not specified); image transferred to *E_ZEKWRITE* in the field *wtinerr* of the structure *xio_wt* (see WAIT command in the Section "iocs_wait waiting for the end of routines").
Read ZEKOSA data	30	no error
	31	Length Redundant Character error
	32	parity error
	33	no data read
	34	no start character
	35	incorrect flux change
	36	no stop character
	37	break in magnetic strip

Format of the data structure *xio_drstat*:

```
struct xio_drstat
{
unsigned char dronl;   /* online/offline    */
unsigned char drdin;   /* voucher status    */
unsigned char unused;  /* free              */
unsigned char doc;     /* extended status   */
unsigned char free1;   /* free              */
unsigned char free2;   /* free              */
unsigned char free3;   /* free              */
};
```

dronl

One of the following values is reported in this field:

S_ONL□ Device is online□
S_OFFL Device is offline

drdin

One of the following values is reported in this field:

S_NOD□ No voucher in position□
S_DIN□ Voucher in position□
S_OLDD Voucher not removed

doc

Internal device errors are reported in this field (if necessary, inform Siemens Nixdorf Service):

Command	**No.**	**Meaning**
Read booking line/footer	>30	undefined
Read OCR data	>30	undefined
Line Finding	30	no error
	31	no line marking/bad line marking
	>31	not specified; image transferred in messages in the field *wtinerr* of the structure *xio_wt* (see WAIT command in the Section "iocs_wait waiting for the end of routines").

If only the normal printer status of the ND90 is to be read the length of the data structure can be specified in the field *datl* of the argument structure *xio_arg*, e.g. sizeof(struct xio_prstat).

If the ZEKOSA status and/or the OCR status of the ND90 must also be read, the maximum data buffer length must be specified in the field *datl* of the argument structure *xio_arg*, e.g. sizeof(struct xio_stnd90).

Any change in the device status is noted in the field *wtst* in the structure *xio_wt* for each WAIT command.

Example of a function that queries the status of a printer:

```
□#include <xio.h>
#include "xio_struct.h"
int bpt_status(syncflag)
int syncflag;          /* routine synchronization flag */
```

```
{
extern int bpt_wait();
extern struct xio_wt order;
extern struct xio_stnd90 nd90stat;
struct xio_stnd90 *nd90_ptr;
extern struct xio_arg argument;
int funct_val, device_no;
int command = X_GTST;
 /* initialization of the argument structure for reading the device status */
nd90_ptr = &nd90stat;
argument.datl = sizeof(nd90stat);
argument.nitems = 0;
argument.dat_buf = (char *)nd90_ptr;
device_no = order.wtdb;  /* logical device no. (see OPEN) */
funct_val = iocs_cntl(device_no, command, &argument);
if (funct_val < 0)
{
/* error in CNTL call */
}
else
{
if (syncflag == X_NOSYNC)
{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* Evaluation of the structure xio_wt;       */
/* the structure xio_wt can only be evaluated */
/* if wtrev != 0 (see iocs_wait)             */
}
}
}
return (funct_val);
}
```

4.3.7.3 Canceling routines

This function is used to cancel PUT commands issued by the application. All routines not yet processed are deleted. Any routines currently being processed are aborted.

If an error occurs while a PUT command is being processed, any remaining routines issued by the application for the device in question are stopped. Canceling the remaining routines allows appropriate reaction to an error.

The CNTL command need not be acknowledged with a WAIT command.

However, during asynchronous processing, canceled routines must be fetched with a WAIT command. The WAIT command tells the application which routines have been canceled. Any data read is transferred to the application. During synchronous processing the CNTL command has no effect.

Example of a function that cancels a routine:

```
#include <xio.h>
#include "xio_struct.h"
int bpt_storno()
```

```
{
extern struct xio_wt order;
extern struct xio_arg argument;
int funct_val, device_no;
/* initialization of the argument structure for canceling routines */
argument.datl =0;
argument.nitems = 0;
argument.dat_buf = (char *)NULL;
device_no = order.wtdb;
funct_val = iocs_cntl(device_no, X_OUTPCAN, &argument);
if (funct_val < 0)
{
/* error in CNTL routine */
}
return (funct_val);
}
```

4.3.7.4 Monitoring print routines

This function is used to check whether or not a PUT command acknowledged by a WAIT command has been executed correctly.

The maximum time the CHECK command should wait for the correct execution of print routines is specified in seconds (< 32,400) in the field *nitems* of the argument structure *xio_arg*. If *nitems* is set to "0" the timer is not started and the CHECK command waits indefinitely if an error occurs.

During asynchronous processing, all remaining print routines must be fetched with a WAIT command prior to the CNTL command.

The call should not be used in connection with SAS printers, since a logical protocol has been defined for communication between IOCS and this type of device. In this case, the call always produces a positive result because all the data is output and acknowledged after a positive WAIT command.

If print routines are not executed correctly within the defined time, the error "Device not ready" is reported. The field *nitems* should therefore be set to a sufficiently high value.

Example of a function that monitors print routines:

```
#include <xio.h>
#include "xio_struct.h"
int bpt_chk(syncflag)
int syncflag;
{
extern int bpt_wait();
extern struct xio_wt order;
extern struct xio_arg argument,
int funct_val, device_no;
int command = X_CHECK;
/* initialization of the argument structure used to monitor routines */
argument.datl =0;
argument.nitems = 30;
argument.dat_buf = (char *)NULL;
device_no = order.wtdb;  /* logical device no. (see OPEN) */
funct_val = iocs_cntl(device_no, command, &argument);
if (funct_val < 0)
{
/* error in CNTL call */
}
else
{
```

```
if (syncflag == X_NOSYNC)
{
funct_val = bpt_wait();
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
/* Evaluation of the structure xio_wt;       */
/* the structure xio_wt can only be evaluated */
/* if wtrev != 0 (see iocs_wait)             */
}
}
}
return (funct_val);
}
```

4.3.8 iocs_wait – waiting for the end of routines

Synopsis

```
#include <xio.h>
int iocs_wait(pollfds, nitems, timeout)
struct xio_wt *pollfds[ ];
int nitems;
int timeout;
```

Description

The WAIT command is used by the application to wait for the completion of one or more IOCS commands. In asynchronous processing, the WAIT command must be issued after each IOCS command with the exception of CLOSE and CNTL (*X_OUTPCAN*).

The value "0" is returned if the routine is accepted, and the structure *xio_wt* is then filled accordingly. When acknowledgments are received from routines reading data, the read buffer is filled and the number of characters read is returned. If the routine is invalid, an appropriate error number (< 0) is returned.

If an application issues several IOCS commands to different devices before it issues a WAIT command, it can either send a "multiple wait" or a WAIT for a specific peripheral. A "multiple wait" is acknowledged when at least one of the routines issued has been completed, or the WAIT has timed out. New WAIT commands must be started for each of the routines not yet executed. A "multiple wait" allows the application to wait for alternating input and output from several devices.

In synchronous processing, a WAIT command always runs implicitly with a timeout period of "-1". When working in this mode, the application must not issue a WAIT command.

nitems
: The number of devices to be awaited is specified in this parameter.

timeout
: A maximum waiting time in seconds is specified for the WAIT command in the parameter *timeout*. If a timeout time of "-1" is set, the application goes 'on hold' until one of the desired routines ends or an error occurs. If a waiting time of zero seconds is set, the WAIT command operates as a check command. If none of the routines finishes in the time set, the WAIT command is acknowledged with a timeout error. The WAIT command must then be restarted if necessary.

  If a print routine is issued to an offline SAS printer, the error *E_TIME* is returned after 180 seconds and the status field *pronl* in the structure *xio_prstat* is set to *S_OFFL*. If the printer switches to *ONLINE* again during this period, the routine is executed as normal and the status *S_ONL* is returned to the application.

pollfds

This parameter transfers an array of pointers that indicate structures where the awaited routines are specified. These structures have the following format:

```
struct xio_wt
{
□□int wtdb; □□□□□□□□□□□□□/* logical device number      */
□□short wtev;□□□□□□□□□□□ /* not interpreted at present */
□□short wtrev;□□□□□□□□□ /* ID of reported routines    */
□□short wtinps;□□□□□□□□ /* seq.no. of last GET/PUT    */
□□short wtouts;□□□□□□□□ /* not interpreted at present */
□□short wtinerr;□□□□□□□ /* error no. of last GET      */
□□short wtouterr;□□□□□□ /* error no. of last PUT      */
□□short wtst;□□□□□□□□□□ /* marker for status changes  */
□□unsigned short wtstsize; /* length of status buffer */
□□union xio_stat *wtpst;□□/* pointer to status buffer */
};
```

wtdb — The logical device number returned by *iocs_open* is transferred in this field.

wtrev — The routine for which the structure was filled is indicated to the application in this field. This field can contain any of the following values:

XIO_NREADY	The structure cannot be interpreted because none of the routines started has been reported as finished. This value is only returned if the timeout period is not "-1". A new WAIT command must be started, or the routine in question must be canceled.
XIO_OPEN	Acknowledgment of an OPEN routine.
XIO_PUT	Acknowledgment of a PUT routine.
XIO_GET	Acknowledgment of a GET routine.
XIO_LOCK	Acknowledgment of a LOCK routine.
XIO_UNLOCK	Acknowledgment of an UNLOCK routine.
XIO_CID	Acknowledgment of a CNTL routine (*GTID*).
XIO_CSTAT	Acknowledgment of a CNTL routine (*GTST*).
XIO_CHECK	Acknowledgment of a CNTL routine (*CHECK*).

wtinps — The sequence number of the last PUT or GET command processed is returned in this field.

wtinerr — Device errors are reported in this field. If no errors occur, the value "0" is returned. The field can only be interpreted if routines have been acknowledged (*wtrev* unequal 0).

The field *wtinerr* can be set to any of the following values (see also Section "Error messages and user reactions"):

E_NOOPEN□ *E_ISLOCK*□ *E_NOLOCK*□

E_COM□	*E_NOFUN*□	*E_FALFUN*□
E_STORNO□	*E_DEV01*□	*E_NOSVR*□
E_SVRNOF□	*E_DEVNOP*□	*E_PROT*□
E_HDW□	*E_ESC*□	*E_NODEV*□
E_PAEND□	*E_COEND*□	*E_COVER*□
E_DEV02□	*E_TIME*□	*E_NOPAPER*□
E_NODEVSVR□	*E_READ*□	*E_DEVESC*□
E_PRIESC□	*E_CHECK*□	*E_UDEVERR*□
E_PRTO□	*E_ZEKTO*□	*E_ZEKLWA*□
E_ZEKWRITE□	*E_ZEKREAD*	*E_DRTO*
E_DRREAD		

wtouterr If an *E_NOLOCK* error is reported in *wtinerr* and a LOCK command is issued for several devices, additional information is entered in this field, indicating whether a device was already locked (*wtouterr* = *E_ISLOCK*). If the device is not locked, the value "0" is entered in *wtouterr*. If a LOCK command is issued for several devices, none of the devices is locked if any is already locked. In this case, *E_NOLOCK* is entered in *wtinerr* for all the devices. The application can determine which of the devices was already locked by interpreting the contents of the field *wtouterr*.

wtst This field indicates whether the device status has changed:

X_OLDST No change in status.

X_NEWST Status has changed. The current status can be read from the status buffer specified.

wtstsize The length of the status buffer transferred must be specified in this field. The length is usually set using the C operator *sizeof*.

wtpst The pointer to the device-specific status buffer must be transferred in this field. If the NULL pointer is transferred in this field, only the status marker *wtst* is updated. In this case, the application can read the current status using the CNTL command "Read device status". The format of the status buffer *pr_stat* is described in the subsection on the CNTL command (see the Section "iocs_cntl control function for printers").

Errors

The WAIT command is not processed if any of the following conditions applies:

E_DEVNOP
: The device has not been opened.

E_XIOSTOP
: The IOCS has been stopped.

E_IO1
: General IOCS error.

E_COM

Error in communication between the processes involved in IOCS.

E_FALSEF
The IOCS function call is invalid.

Example

```
#include <xio.h>
#include "xio_struct.h"
int bpt_wait()
{
extern struct xio_wt order;
struct xio_wt *order_ptr;
extern struct xio_stnd90 nd90stat;
int funct_val;
int number = 1;
int timeout = 1000;
/* initialization of the routine structure used to wait for the end of a routine */
order.wtstsize = sizeof(struct xio_stnd90);
order.wtpst = &nd90stat;
order_ptr = &order;
funct_val = iocs_wait(&order_ptr, number, timeout);
if (funct_val < 0)
{
/* error in WAIT call */
}
else
{
if (order.wtinerr < 0)  /* device error */
{
funct_val = order.wtinerr;
fprintf(stderr, "Error has occurred on device with log.
number %d\n", order.wtdb);
}
}
return(funct_val);
}
```

4.4 Special control sequences for the ND90

This subsection describes the special control sequences that are supported by the ND90. but do not belong to the set of compatible printer control sequences.

4.4.1 LEDH/LEDF – setting voucher header/footer

Synopsis

LEDH:

ASCII:	ESC	[	=	<	2	1	;	0	;	s
Hexadecimal:	1B	5B	3D	3C	32	31	3B	30	3B	73

LEDF:

ASCII:	ESC	[	=	<	2	1	;	1	;	s

Hexadecimal:	1B	5B	3D	3C	32	31	3B	31	3B	73

Description

LEDH and *LEDF* can be used to select the reference border for passbook or voucher processing.

LEDH sets the voucher header as the reference border.

LEDF sets the voucher footer as the reference border.

4.4.2 SPBM – setting passbook parameters

Synopsis

ASCII:	ESC	[	=	!	*n1*	;	*n2*	;	...	*n13*	;	1
Hexadecimal:	1B	5B	3D	21	*n1*	3B	*n2*	3B	...	*n13*	3B	31

Description

SPBM can be used to set the passbook parameters.The parameters *n1* through *n13* correspond to the fields *prfpar1* through *prfpar13* in Section "Setting passbook parameters".

4.4.3 SCLP/SCLP_C – switching coding line processing on/off

Synopsis

SCLP:

ASCII:	ESC	[	=	<	1	8	;	1	;	s
Hexadecimal:	1B	5B	3D	3C	31	38	3B	31	3B	73

SCLP_C:

ASCII:	ESC	[	=	<	1	8	;	0	;	s
Hexadecimal:	1B	5B	3D	3C	31	38	3B	30	3B	73

Description

SCLP switches coding line processing on.

SCLP_C switches coding line processing off.

4.4.4 SPBH/SDOH – setting passbook processing/single sheet processing

Synopsis

SPBH:

ASCII:	ESC	[	=	<	1	9	;	0	;	s
Hexadecimal:	1B	5B	3D	3C	31	39	3B	30	3B	73

SDOH:

ASCII:	ESC	[	=	<	1	9	;	1	;	s
Hexadecimal:	1B	5B	3D	3C	31	39	3B	31	3B	73

Description

SPBH switches to passbook processing (default setting). This sequence is necessary to ensure normal paper feeding of thicker documents such as passbooks, and to prevent damage to the print head.

SDOH switches to single-sheet processing.

4.4.5 SROB/SROA – setting OCR-B/OCR-A character set for read functions

Synopsis

SROB:

ASCII:	ESC	[	=	<	5	1	;	1	;	s
Hexadecimal:	1B	5B	3D	3C	35	31	3B	30	3B	73

SROA:

ASCII:	ESC	[	=	<	5	1	;	2	;	s
Hexadecimal:	1B	5B	3D	3C	35	31	3B	32	3B	73

Description

These sequences are used to set the corresponding OCR character set for all OCR read functions (line finding, reading OCR data, reading posting lines/footers).

SROB activates the OCR-B character set (default setting).

SROA activates the OCR-A character set.

4.4.6 SWOB/SWOA – setting the OCR-B/OCR-A character set for data output

Synopsis

SWOB:

ASCII:	ESC	[	=	<	5	0	;	1	;	s
Hexadecimal:	1B	5B	3D	3C	35	30	3B	31	3B	73

SWOA:

ASCII:	ESC	[	=	<	5	0	;	2	;	s
Hexadecimal:	1B	5B	3D	3C	35	30	3B	32	3B	73

Description

These sequences are used to set the corresponding OCR character set for data output (see also the Section "SWON setting the G0 character set for data output").

SWOB activates the OCR-B character set.

SWOA activates the OCR-A character set.

4.4.7 SWON – setting the G0 character set for data output

Synopsis

SWON:

ASCII:	ESC	[	=	<	5	0	;	0	;	s
Hexadecimal:	1B	5B	3D	3C	35	30	3B	30	3B	73

Description

SWON switches from *SWOB/SWOA* back to the normal GO character set (default setting).

4.5 Error messages and user reactions

4.5.1 General error messages

Error	No.	Cause	Reaction
E_MFILE	-1	Too many devices have been opened (*IO_DEVNCNT* in *iocs_config.h*)	Some devices must be connected to another system, if all devices are to be used simultaneously
E_NOXIO	-2	IOCS has not been started	Start IOCS
E_NOENT	-3	Required configuration unavailable:	
		$IOCS not or wrong set	Set *$IOCS*
		iocs.dev, *iocs.tty* or *iocs.fmod* not found	Set *$IOCS*, restore files, check access rights
		Application has no access right	Check entry $ACC in iocs.dev
		Specified number of devices to be operated is too large for the device control process (*IO_MAXDEV* in *iocs_config.h*)	Specify a lower number
		Invalid character table configured for the device control process	Evaluate the process trace, in order to determine tables
		The device control process has terminated for an unknown reason	Programming error, evaluate process trace, rectify error
		The device control process was deleted by the system administrator	

		Invalid paramenters for OPEN	Evaluate the process trace, rectify error
E_ISOPEN	-4	Device is already open	Check application
E_NOOPEN	-5	Unable to open device:	Check device or analyse logging in more detail
		Device switched off	Switch on device
		Device configured incorrectly	Check configuration
		Error in one of the files *iocs.dev*, *iocs.tty*, or *iocs.fmod*	Call *iocs_config -c*, check visually
		Internal error in the device control process	Evaluate process trace
		Emulation table could not be read	Call *iocs_config -c*, check visually
		Device already locked by another process (e.g. spool)	Check system configuration
		Unknown device ID read for log printer	Check configuration (it is possible that the device cannot be operated via IOCS)
E_FALSEF	-6	Invalid function call	Check sequence of routines
E_ISLOCK	-7	Device is already locked	Wait and execute LOCK routine again
E_MSEQ	-8	Maximum sequence number reached	Execute WAIT routine
E_XIOSTOP	-9	IOCS has been stopped	Restart IOCS
E_IO1	-10	General IOCS error 1:	
		The file *$IOCS/fifos/filename* could not be created	Check access rights
		The central process could not switch to *$IOCS_TRACE*	Check existence of directory and/or access rights
		The *SIGALRM* signal could not be set	Program error, notify SNI Service Center
		The central process could not open the *iocs.dev* file	Check access rights to *$IOCS*
		The device control process could not be started	Check existence of device control process and/or execution rights; process table full or error with *fork()*
		The file *$IOCS/fifos/filename*	The device control

		for the device control process could not be created or opened	process terminated beforehand (see *E_NOENT*); check existence of file and/or access rights
E_IO2	-11	General IOCS error 2	Terminate IOCS and start under root
E_NOLOCK	-12	Device is not locked	Check application
E_STORNO	-13	Routine has been canceled	No reaction; positive response to *STORNO*
E_NOMEM	-14	No storage space available	Increase system resources
E_NFILE	-15	Maximum number of users for device	Check configuration (*IO_MODNCNT* in *iocs_config.h*)
E_COM	-16	Error in communication between IOCS processes:	
		Application or device control process could not open the file *$IOCS/fifos/filename*	Check existence of *$IOCS/fifos* directory and check the file and/or access rights
		Write (read) error in the file *$IOCS/fifos/filename*; process is no longer active	Determine cause of crash
E_TOOBIG	-17	Too much data for PUT/GET routine	Check application
E_DEVNOP	-18	Device was not opened beforehand	Check application
E_INKONS	-19	New IOCS configuration is inconsistent	Check configuration
E_OPTRACE	-20	Process trace file could not be opened	Check access rights
E_NOTRACE	-21	Process trace is not active for this process	Check process id
E_ISTRACE	-22	Process trace already active for this process	Check process id
E_NOIOCSP	-23	IOCS does not recognize the process	Check process id
E_NONEWP	-24	Process release is less than V3.0	To switch on the process trace in the IOCS application process, the application must be linked again to either of the libraries *libxio_t.a* or *libxio_t.so*. When the application is restarted, a PAV file called

			lu.user_id.process_id is created in the *$IOCS/tmp/iocs* directory. To switch off the process trace in the IOCS application process, the application must be linked again to either of the libraries *libxio.a* or *libxio.so*. To switch on the process trace in the device control process, the corresponding device control process must be entered in the *iocs.dev* file with a *.t* extension, and the central process must be stopped and restarted. After the central process is started, a PAV file called *lg.process_id* is created for the relevant device in the *$IOCS/tmp/iocs* directory. To switch off the process trace in the device control process, the corresponding device control process must be entered in the file *iocs.dev* without the *.t* extension, and the central process must be stopped and restarted.
E_INIOCS	-25	Internal programming error	Notify SNI Service Center
E_NEWST	-41	Change of status	Check status
E_NOFUN	-50	Peripheral unable to execute function	Check application
E_FALFUN	-51	Function not implemented in function module	Check application
E_DEV01	-61	Error when accessing a device:	
		Device is no longer ready for operation, e.g. if the operation signal is no longer present with V.24 printers.	Set the device correctly
		A system call for the device is returned with an	Evaluate process trace

		error	
E_NOSVR	-62	Device driver not interated	Check configuration in *iocs.dev*
E_SVRNOF	-63	Invalid function for device driver	Check application
E_DEVNOOP	-64	Device could not be opened	Check device
E_PROT	-65	Protocol error in device	Restart application, notifySNI Service Center
E_HDW	-66	Hardware error	Check application/device
E_ESC	-67	ESC sequence invalid or wrong character found	Evaluate process trace, rectify error in application data
E_NODEV	-68	Device not ready for operation, e.g. printer set to STOP, or error message from printer	Check device and restart application
E_PAEND	-69	Paper out	Add paper
E_COEND	-70	Ribbon out	Replace ribbon
E_COVER	-71	Cover open	Close cover
E_DEV02	-72	General device error	Notify SNI Service Center
E_TIME	-73	Timeout error	Check device/emulation
E_NOPAPER	-74	No voucher	Insert voucher
E_NODEVSVR	-75	Incorrect printer configuration	Change hardware configuration
E_READ	-76	Read error	Repeat read routine if necessary
E_NOCOD	-77	Code table missing	Check configuration in *iocs.dev*
E_NOFLAG	-78	Invalid flag in emulation table	Check emulation table
E_PRINIT	-79	Printer was initialized	Repeat routine
E_DEVESC	-80	Workstation reports "incorrect sequence"	Repeat routine, notify SNI Service Center
E_PRIESC	-81	Printer reports "incorrect sequence"	Repeat routine, notify SNI Service Center
E_CHECK	-82	The wait time expired during the *X_CHECK* job	Check device, restart application, increase wait time (if necessary)
E_UDEVERR	-83	Workstation reports undefined error code	Repeat routine, notify SNI Service Center

4.5.2 Error messages specific to the ND90

Error	No.	Cause	Reaction

		Timeout error	Check device/emulation
E_PRTO	-73	– Printer	
E_ZEKTO	-73	– ZEKOSA	
E_DRTO	-73	– OCR reader	
E_ZEKLWA	-157	Buffer size exceeds track capacity (ZEKOSA)	Check application
E_ZEKWRITE	-161	Write error (ZEKOSA)	Check magnetic strip
		Read error	
E_ZEKREAD	-162	– ZEKOSA	Check status
E_DRREAD	-162	– OCR reader	Check line marker

5 Compatible control sequences

5.1 Preface

A set of EPSON control sequences has been defined to provide compatible support for various types of printer. These sequences are described as "compatible control sequences". They contain all sequences necessary for normal operation of printers. These sequences are compatible on all printers released by Siemens Nixdorf Informationssysteme AG for this purpose, whereby functions not supported by the hardware (e.g. *BELL* if there isn't a buzzer) are ignored or dealt with logically.

The print control sequences described in this section are based on the Epson industrial standard (ESC/P), supplemented by Siemens Nixdorf Informationssysteme AG-defined sequences (NX/P). The sequences are addressed and programmed in abbreviated form, except *ABSPOS, HT_SET, RELPOS, SWCCC* and *VT_SET*. If the abbreviations are to include variables, they are enclosed in brackets and separated by commas. Example: *S_TOP(24)* must be specified to set the start of page to 1 inch.

5.1.1 Purpose

This section of the manual provides a detailed description of the compatible control sequences used to program printers via IOCS.

5.1.2 Layout of section

The Section "Overview of control sequence functions" indicates the functions of all control sequences.

The Section "Overview of control sequence encoding" indicates the codes of all control sequences.

The Section "Individual description of control sequences" contains a detailed description of all control sequences that are supported regardless of the respective printer used.

The Section "Control sequences of the individual printers" lists the control sequences supported by the respective printers.

5.2 Overview of control sequence functions

Abbreviation	Function	NX/P
ABSPOS	Set absolute dot position	no
AFEN	Automatic page feed on	no
AFEN_C	Automac page feed off	no
BELL	Acoustic signal	no
BMARGIN	Set lower margin	no
BMARGIN_C	Delete lower margin	no
BPM	Bold type on	no
BPM_C	Bold type off	no
BS	Backspace	no
COLOUR	Select print colour	no
CPI_10	10 characters per inch	no
CPI_12	12 characters per inch	no
CPI_15	15 characters per inch	no
CR	Carriage return	no
CURSIV	Italics on	no
CURSIV_C	Italics off	no

DCHH	Double character height on	no
DCHH_C	Double character height off	no
DRAFT	Draft mode on	no
EPM	Double character width on	no
EPM_C	Double character width off	no
FEEDER 1	Select tray 1	no
FEEDER 2	Select tray 2	no
FEJ	Form/page eject	no
FF	Form/page feed	no
FONT	Select font	no
HT	Horizontal tabulation	no
HT_SET	Set horizontal tabulator at position n	no
LF	Line feed	no
LF_LINES	Line feed n lines	yes
LFB	Line feed backwards	no
LPI_3	3 lines per inch	yes
LPI_5	5 lines per inch	yes
LPI_6	6 lines per inch	no
LPI_8	8 lines per inch	no
LPI_60	60 lines per inch	no
LQ	Letter quality (LQ) on	no
LS0	Set character table G0 to GL until next changeover	yes
LS1	Set character table G1 to GL until next changeover	yes
LS2	Set character table G2 to GL until next changeover	yes
LS3	Set character table G3 to GL until next changeover	yes
LS1R	Set character table G1 to GR until next changeover	yes
LS2R	Set character table G2 to GR until next changeover	yes
LS3R	Set character table G3 to GR until next changeover	yes
NLQ	Near letter quality (NLQ) on	no
PLENGTH	Page length in lines	no
PROPORT	Proportional spacing on	no
PROPORT_C	Proportional spacing off	no
RELPOS	Set relative dot position	no
RESET	Printer initialization	no
SAN	Switch to print station n	yes
SI	Condensed print on	no

SI_C	Condensed print off	no
SLM	Set left margin	no
SLM_60	Set left margin to 1/60 inches	yes
SLOW	Half speed on	no
SLOW_C	Half speed off	no
S_TOP	Set start of page	yes
SPM	Shadow printing on	no
SPM_C	Shadow printing off	no
SRM	Set right margin	no
SS2	Set character table G2	yes
SS3	Set character table G3	yes
STYLE	Select masterstyle	no
SUBSCRIPT	Subscript on	no
SUPERSCRIPT	Superscript on	no
SUBP_C	Super/subscript off	no
SWCCC	Select control character class	yes
SWCTAB	Select code table	yes
UL	Underlining on	no
UL_C	Underlining off	no
UNIDIR	Unidirectional print on	no
UNIDIR_C	Unidirectional print off	no
VT	Vertical tabulation	no
VT_SET	Set vertikal tabulator at line *n*	no

5.3 Overview of control sequence encoding

Abbreviation	ASCII code	Decimal code	Hexadecimal code
ABSPOS	ESC $ *n1 n2*	27 36 *n1 n2*	1B 24 *n1 n2*
AFEN	ESC EM 4	27 25 52	1B 19 34
AFEN_C	ESC EM 0	27 25 48	1B 19 30
BELL	BEL	7	07
BMARGIN	ESC N *n*	27 78 *n*	1B 4E *n*
BMARGIN_C	ESC O	27 79	1B 4F
BPM	ESC E	27 69	1B 45
BPM_C	ESC F	27 70	1B 46
BS	BS	8	08
COLOUR	ESC r *n*	27 114 *n*	1B 72 *n*
CPI_10	ESC P	27 80	1B 50

CPI_12	ESC M	27 77	1B 4D
CPI_15	ESC g	27 103	1B 67
CR	CR	13	0D
CURSIV	ESC 4	27 52	1B 34
CURSIV_C	ESC 5	27 53	1B 35
DCHH	ESC w 1	27 119 49	1B 77 31
DCHH_C	ESC w 0	27 119 48	1B 77 30
DRAFT	ESC x 0	27 120 48	1B 78 30
EPM	ESC W 1	27 87 49	1B 57 31
EPM_C	ESC W 0	27 87 48	1B 57 30
FEEDER 1	ESC EM 1	27 25 49	1B 19 31
FEEDER 2	ESC EM 2	27 25 50	1B 19 32
FEJ	ESC EM R	27 25 82	1B 19 52
FF	FF	12	0C
FONT	ESC k *n*	27 107 *n*	1B 6B *n*
HT	HT	9	09
HT_SET	ESC D *n1 n2 ... n32* NUL	27 68 *n1 n2 ... n32* 00	1B 44 *n1 n2 ... n32* 00
LF	LF	10	0A
LF_LINES	ESC [= < 2 7 ; *n* ; s	27 91 61 60 50 55 59 *n* 59 115	1B 5B 3D 3C 32 37 3B *n* 3B 73
LFB	ESC j	27 106	1B 6A
LPI_3	ESC A DC4	27 65 20	1B 41 14
LPI_5	ESC A FF	27 65 12	1B 41 0C
LPI_6	ESC 2	27 50	1B 32
LPI_8	ESC 0	27 48	1B 30
LPI_60	ESC A SOH	27 65 01	1B 41 01
LQ	ESC x 1	27 120 49	1B 78 31
LS0	SI	15	0F
LS1	SO	14	0E
LS2	ESC n	27 110	1B 6E
LS3	ESC o	27 111	1B 6F
LS1R	ESC ~	27 126	1B 7E
LS2R	ESC }	27 125	1B 7D
LS3R	ESC |	27 124	1B 7C
NLQ	ESC x 1	27 120 49	1B 78 31
PLENGTH	ESC C *n*	27 67 *n*	1B 43 *n*
PROPORT	ESC p 1	27 112 49	1B 70 31

PROPORT_C	ESC p 0	27 112 48	1B 70 30
RELPOS	ESC \ *n1 n2*	27 92 *n1 n2*	1B 5C *n1 n2*
RESET	ESC @	27 64	1B 40
SAN	ESC [= < 3 5 ; *n* ; s	27 91 61 60 51 53 59 *n* 59 115	1B 5B 3D 3C 33 35 3B *n* 3B 73
SI	ESC SI	27 15	1B 0F
SI_C	DC2	18	12
SLM	ESC l *n*	27 108 *n*	1B 6C *n*
SLM_60	ESC [= < 7 ; *n* ; s	27 91 61 60 55 59 *n* 59 115	1B 5B 3D 3C 37 3B *n* 3B 73
SLOW	ESC s 1	27 115 49	1B 73 31
SLOW-C	ESC s 0	27 115 48	1B 73 30
S_TOP	ESC [= < 2 2 ; *n* ; s	27 91 61 60 50 50 59 *n* 59 115	1B 5B 3D 3C 32 32 3B *n* 3B 73
SPM	ESC G	27 71	1B 47
SPM_C	ESC H	27 72	1B 48
SRM	ESC Q *n*	27 81 *n*	1B 51 *n*
SS2	(non-displayable)	142	8E
SS3	(non-displayable)	143	8F
STYLE	ESC ! *n*	27 33 *n*	1B 21 *n*
SUBSCRIPT	ESC S 1	27 83 49	1B 53 31
SUPERSCRIPT	ESC S 0	27 83 48	1B 53 30
SUBP_C	ESC T	27 84	1B 54
SWCCC	ESC [= < 9 9 ; *n1* ; *n2* s	27 91 61 60 57 57 59 *n1* 59 *n2* 115	1B 5B 3D 3C 39 39 3B *n1* 3B *n2* 73
SWCTAB	ESC [= < 9 8 ; *n* ; s	27 91 61 60 57 56 59 *n* 59 115	1B 5B 3D 3C 39 38 3B *n* 3B 73
UL	ESC - 1	27 45 49	1B 2D 31
UL_C	ESC - 0	27 45 48	1B 2D 30
UNIDIR	ESC U 1	27 85 49	1B 55 31
UNIDIR-C	ESC U 0	27 85 48	1B 55 30
VT	VT	11	0B
VT_SET	ESC B *n1 n2 ... n16* NUL	27 66 *n1 n2 ... n16* 00	1B 42 *n1 n2 ... n16* 00

5.4 Individual description of control sequences

5.4.1 ABSPOS – set absolute dot position

Synopsis

ASCII:	ESC	$	*n1*	*n2*

Decimal:	27	36	*n1*	*n2*
Hexadecimal:	1B	24	*n1*	*n2*

Description

ABSPOS places the print head at an absolute dot position, which is determined with the formula "*n2* * 256 + *n1*", irrespective of the actual character height. Each dot corresponds to 1/60 inches. For *n1*, select a value between "0" and "255", and for *n2* a value between "0" and "3", where "816" can be specified as the maximum dot position.

Note

ABSPOS is ignored if the specified position extends beyond the right-hand margin.

See also

RELPOS (Section "RELPOS set relative dot position")

5.4.2 AFEN/AFEN_C – automatic page feed on/off

Synopsis

AFEN:

ASCII:	ESC	EM	4
Decimal:	27	25	52
Hexadecimal:	1B	19	34

AFEN_C:

ASCII:	ESC	EM	0
Decimal:	27	25	48
Hexadecimal:	1B	19	30

Description

AFEN switches on automatic page feed in printers with feeders. The tray to be used for feeding is then selected.

AFEN_C switches on automatic page feed in printers with feeders.

Note

When automatic page feed is on, a page/form is fed in automatically with *FF*.

The setting *AFEN* remains valid until switched off with *AFEN_C* or until standard values are set with *RESET* or a CLOSE/OPEN routine.

See also

FEEDER 1, *FEEDER 2* (Section "FEEDER1/FEEDER2 select tray 1/tray 2")

5.4.3 BELL – acoustic signal

Synopsis

ASCII:	BEL
Decimal:	7
Hexadecimal:	07

Description

BELL activates the acoustic printer signal.

5.4.4 BMARGIN/BMARGIN_C – set/delete lower margin

Synopsis

BMARGIN:

BMARGIN_C:

ASCII:	ESC	N	*n*
Decimal:	27	78	*n*
Hexadecimal:	1B	4E	*n*

ASCII:	ESC	O
Decimal:	27	79
Hexadecimal:	1B	4F

Description

BMARGIN sets a lower margin of *n* lines in the current line spacing. For *n*, select a value between "0" and "127", which can be entered using control characters. The actual length of a page depends on the line spacing defined.

BMARGIN_C deletes the lower margin determined with *BMARGIN*, so that the perforation is used for printing if the software does not assume control of the page format.

5.4.5 BPM/BPM_C – bold type on/off

Synopsis

BPM:

ASCII:	ESC	E
Decimal:	27	69
Hexadecimal:	1B	45

BPM_C:

ASCII:	ESC	O
Decimal:	27	70
Hexadecimal:	1B	46

Description

BPM switches on bold type and can occur at any position within a print line. All characters are printed in bold from this position.

BPM_C switches off bold type and can occur at any position within a print line.

Note

BPM can extend beyond the line and can only be switched off again with *BPM_C* or *RESET*.

See also

SPM (Section "SPM/SPM_C shadow printing on/off")

5.4.6 BS – backspace

Synopsis

ASCII:	BS
Decimal:	8
Hexadecimal:	08

Description

BS prints the data in the print buffer and then moves the print head one character to the left, so that characters can be overtyped.

Note

If the print head is positioned on the left margin, if the previous character is an *HT* character or if a command specifying an absolute or relative dot position was issued, the BS command is ignored.

5.4.7 COLOUR – select print colour

Synopsis

ASCII:	ESC	r	*n*
Decimal:	27	114	*n*

Hexadecimal:	1B	72	*n*

Description

COLOUR selects one of the following print colours: (*n* must be coded binary):

0 = black	1 = red	2 = blue	3 = violet
4 = yellow	5 = orange	6 = green	

Note

If the printer only has a black ribbon, the colour selection is ignored and only black is printed.

5.4.8 CPI_10/CPI_12/CPI_15 – 10/12/15 characters per inch

Synopsis

CPI_15:

ASCII:	ESC	g
Decimal:	27	103
Hexadecimal:	1B	67

CPI_10:

ASCII:	ESC	P
Decimal:	27	80
Hexadecimal:	1B	50

CIP_12:

ASCII:	ESC	M
Decimal:	27	77
Hexadecimal:	1B	4D

Description

CPI switches the printer to a character spacing of 10 (*CPI_10*), 12 (*CPI_12*) or 15 (*CPI_15*) characters per inch, and can occur at any position within a print line. The setting is made for the specified number of print stations.

Note

The setting is retained until it is switched by the next *CPI*, or until the default settings are restored by *RESET* or a CLOSE/OPEN routine.

RESET or a CLOSE/OPEN routine switches automatically to *CPI_10*.

5.4.9 CR – carriage return

Synopsis

ASCII:	CR
Decimal:	13
Hexadecimal:	0D

Description

CR positions the print head at column "0" and can occur at any position within a print line.There is no line feed. The actual line will be printed.

Note

In printers where *CR = CR + LF* is set with the coding switch, *CR* generates a carriage return with a line feed.

5.4.10 CURSIV/CURSIV_C – italics on/off

Synopsis

CURSIV:

ASCII:	ESC	4
Decimal:	27	52
Hexadecimal:	1B	34

CURSIV_C:

ASCII:	ESC	5
Decimal:	27	53
Hexadecimal:	1B	35

Description

CURSIV switches on italic type for all subsequent characters.

CURSIV_C switches off the italic type switched on with *CURSIV*.

5.4.11 DCHH/DCHH_C – double character height on/off

Synopsis

DCHH:

ASCII:	ESC	w	1
Decimal:	27	119	49
Hexadecimal:	1B	77	31

DCHH_C:

ASCII:	ESC	w	0
Decimal:	27	119	48
Hexadecimal:	1B	77	30

Description

DCHH switches on double character height. All characters printed subsequently are twice the usual height.

DCHH_C switches to single character height, i.e. *DCHH* is switched off.

Note

DCHH cannot be activated at the same time as *SUPERSCRIPT*, *SUBSCRIPT* or *SI*.

If the line spacing is less than 24/180 inches, double height characters cannot be printed; with a normal spacing of 1/6 inches, they will overlap. *DCHH* should therefore always be set to at least 44/180 inches.

See also

SUBSCRIPT (Section "SUBSCRIPT subscript on")

SUPERSCRIPT (Section "SUPERSCRIPT superscript on")

SI (Section "SI/SI_C condensed print on/off")

5.4.12 DRAFT – draft mode

Synopsis

ASCII:	ESC	x	0
Decimal:	27	120	48
Hexadecimal:	1B	78	30

Description

DRAFT switches the printer to draft mode.

See also

LQ (Section "LQ letter quality on")

NLQ (Section "NLQ near letter quality on")

5.4.13 EPM/EPM_C – double character width on/off

Synopsis

EPM:

ASCII:	ESC	W	1
Decimal:	27	87	49
Hexadecimal:	1B	57	31

EPM_C:

ASCII:	ESC	W	0
Decimal:	27	87	48
Hexadecimal:	1B	57	30

Description

EPM switches to double character width, and can occur at any position within a print line. *EPM* can extend beyond the line.

EPM_C switches to single character width, i.e. *EPM* is switched off. *EPM_C* can occur at any position within a print line.

5.4.14 FEEDER1/FEEDER2 – select tray 1/tray 2

Synopsis

FEEDER1:

ASCII:	ESC	EM	1
Decimal:	27	25	49
Hexadecimal:	1B	19	31

FEEDER2:

ASCII:	ESC	EM	2
Decimal:	27	25	50
Hexadecimal:	1B	19	32

Description

FEEDER1/FEEDER2 switches to tray "1" or "2" in printers with feeders. The paper is taken from tray "1"or "2" if feed commands are then entered.

Note

The respective setting remains until a switch is made to another tray, until the automatic page feed is switched off with *AFEN_C*, or until standard values are set with *RESET* or a CLOSE/OPEN routine.

If this function is to run correctly, a switch must have been made beforehand to automatic page feed with the function *AFEN*.

See also

AFEN, AFEN_C (Section "AFEN/AFEN_C automatic page feed on/off")

5.4.15 FEJ – form/page eject

Synopsis

ASCII:	ESC	EM	R
Decimal:	27	25	82
Hexadecimal:	1B	19	52

Description

FEJ is used to eject a form/page in printers with feeders or page feed. There is no automatic page/form eject.

See also

FF (Section "FF form/page feed")

5.4.16 FF – form/page feed

Synopsis

ASCII:	FF
Decimal:	12

Hexadecimal:	0C

Description

In printers with feeders or form feed, *FF* generates a form/page feed. If a page has already been fed in, it is first ejected.

See also

FEJ (Section "FEJ form/page eject")

LF_LINES (Section "LF_LINES line feed n lines")

5.4.17 FONT – select font

Synopsis

ASCII:	ESC	k	*n*
Decimal:	27	107	*n*
Hexadecimal:	1B	6B	*n*

Description

FONT selects one of the two standard *LQ* fonts or the font of a plugin module, as follows (*n* must be coded binary):

0 = Roman	1 = Sans serif	2 = Courier	3 = Prestige	4 = Script

Note

If the printer is in draft mode when this sequence is issued, the changes are only effective when *LQ* mode is selected.

5.4.18 HT – horizontal tabulation

Synopsis

ASCII:	HT
Decimal:	9
Hexadecimal:	09

Description

HT initiates a jump to the next horizontal tabulator and can occur at any position within a print line. The tabulator positions can be skipped by entering *HT* several times.

Note

It is always the next tabulator position that is used. It is not possible to jump backwards. *HT* jumps do not extend beyond the line. There is generally a tabulator position at every 8th column.

See also

HT_SET (Section "HT_SET set horizontal tabulator at position n")

5.4.19 HT_SET – set horizontal tabulator at position n

Synopsis

ASCII:	ESC	D	*n1*	*n2*	...	*n32*	NUL
Decimal:	27	68	*n1*	*n2*	...	*n32*	00
Hexadecimal:	1B	44	*n1*	*n2*	...	*n32*	00

Description

HT_SET sets 1 to 32 horizontal tabs. These are determined by *n1–n32*, where the values lie between "1" and "255" and must be in ascending order. The positions are absolute values, i.e. they always refer to the start of line.

Note

The printable positions begin at column "0". Extending beyond the line is not permitted. The standard horizontal tabs set must first be deleted with the sequence *ESC D NUL*.

See also

HT (Section "HT horizontal tabulation")

5.4.20 LF – line feed

Synopsis

ASCII:	LF
Decimal:	10
Hexadecimal:	0A

Description

LF performs a line feed. It can be issued at any point in a print line. The print head remains at the same column, while the paper is fed forwards one line at the line spacing set.

Note

If *LF = LF + CR* is set in the printer with the coding switch, LF generates a line feed with carriage return.

See also

LF_LINES (Section "LF_LINES line feed n lines")

5.4.21 LF_LINES – line feed n lines

Synopsis

ASCII:	ESC	[	=	<	2	7	;	*n*	;	s
Decimal:	27	91	61	60	50	55	59	*n*	59	115
Hexadecimal:	1B	5B	3D	3C	32	37	3B	*n*	3B	73

Description

LF_LINES performs a line feed. It can be issued at any point in a print line. The print head remains at the same column, while the paper is fed forwards n lines at the line spacing set.

See also

LF (Section "LF line feed")

5.4.22 LFB – line feed backwards

Synopsis

ASCII:	ESC	j
Decimal:	27	106
Hexadecimal:	1B	6A

Description

LFB forces a backwards line feed.

Note

This function is only possible on printers which have the suitable mechanisms.

5.4.23 LPI_3/LPI_5/LPI_6/LPI_8/LPI_60 – 3/5/6/8/60 lines per inch

Synopsis

LPI_3:

ASCII:	ESC	A	DC4
Decimal:	27	65	20
Hexadecimal:	1B	41	14

LPI_5:

ASCII:	ESC	A	FF
Decimal:	27	65	12
Hexadecimal:	1B	41	0C

LPI_6:

ASCII:	ESC	2
Decimal:	27	50
Hexadecimal:	1B	32

LPI_8:

ASCII:	ESC	0
Decimal:	27	48
Hexadecimal:	1B	30

LPI_60:

ASCII:	ESC	A	SOH
Decimal:	27	65	01
Hexadecimal:	1B	41	01

Description

LPI switches the printer to a line spacing of 3 (*LPI_3*), 5 (*LPI_5*), 6 (*LPI_6*), 8 (*LPI_8*), or 60 (*LPI_60*) lines per inch. The respective sequence should be programmed after *CR* and before the first printable character. The next line feed observes the programmed spacing. The setting is made for the specified number of print stations.

Note

The setting is retained until it is switched by the next *LPI*, or until the default settings are restored by *RESET* or a CLOSE/OPEN routine.

RESET or a CLOSE/OPEN routine select *LPI_6* by default.

5.4.24 LQ – letter quality on

Synopsis

ASCII:	ESC	x	1
Decimal:	27	120	49
Hexadecimal:	1B	78	31

Description

LQ switches on the printer from normal type to letter quality mode. *LQ* should only be issued after *CR* + *LF*.

Note

LQ uses the same print control sequence as *NLQ* and is canceled by *DRAFT*.

See also

DRAFT (Section "DRAFT draft mode")

NLQ (Section "NLQ near letter quality on")

5.4.25 LS0/LS1/LS2/LS3 – set character table G0/G1/G2/G3 to GL until next changeover

Synopsis

LS0:

ASCII:	SI
Decimal:	15
Hexadecimal:	0F

LS1:

ASCII:	SO
Decimal:	14
Hexadecimal:	0E

LS2:

ASCII:	ESC	n
Decimal:	27	110
Hexadecimal:	1B	6E

LS3:

ASCII:	ESC	o
Decimal:	27	111
Hexadecimal:	1B	6F

Description

The characters following the *LS* sequence are generated from the character table G0, G1, G2 or G3 using the code range 20–7F. The setting is retained until a different GL character table is set.

Note

The character table G0 is generally set in GL. The print control functions *LS, LSR* and *SS* can be mixed arbitrarily within a job. The standard code table adopted is marked number "1" in the *iocs.dev* file.

See also

LS1R, LS2R, LS3R (Section "LS1R/LS2R/LS3R set character table G1/G2/G3 to GR until next changeover")

SS2, SS3 (Section "SS2/SS3 set character table G2/G3")

SWCTAB (Section "SWCTAB select code table")

5.4.26 LS1R/LS2R/LS3R – set character table G1/G2/G3 to GR until next changeover

Synopsis

LS1R:

ASCII:	ESC	~
Decimal:	27	126
Hexadecimal:	1B	7E

LS2R:

ASCII:	ESC	}
Decimal:	27	125
Hexadecimal:	1B	7D

LS3R:

ASCII:	ESC	|
Decimal:	27	124
Hexadecimal:	1B	7C

Description

The characters following the *LSR* sequence are generated from the character table G1, G2 or G3 using the code range A0–FF. The setting is retained until a different GR character table is set.

Note

The character table G1 is generally set in GR. The print control functions *LS, LSR* and *SS* can be mixed arbitrarily within a job. The standard code table adopted is marked number "1" in the *iocs.dev* file.

See also

LS0, LS1, LS2, LS3 (Section "LS0/LS1/LS2/LS3 set character table G0/G1/G2/G3 to GL until next changeover")

SS2, SS3 (Section "SS2/SS3 set character table G2/G3")

SWCTAB (Section "SWCTAB select code table")

5.4.27 NLQ – near letter quality on

Synopsis

ASCII:	ESC	x	1
Decimal:	27	120	49
Hexadecimal:	1B	78	31

Description

NLQ switches the printer from normal type to letter quality mode. *NLQ* should only be issued after *CR* + *LF*.

Note

NLQ uses the same print control sequence as *LQ* and is switched off by *DRAFT*.

See also

DRAFT (Section "DRAFT draft mode")

LQ (Section "LQ letter quality on")

5.4.28 PLENGTH – page length in lines

Synopsis

ASCII:	ESC	C	*n*
Decimal:	27	67	*n*
Hexadecimal:	1B	43	*n*

Description

PLENGTH sets the page length to *n* lines in the selected line spacing. For *n*, specify a value between "1" and "127". The actual page length is thus based on the line spacing set.

Note

The start of page position is the line in which the print head is located.

The lower margin determined with *BMARGIN* and the jump established with the mode selection function over the partition are deleted.

See also

BMARGIN (Section "BMARGIN/BMARGIN_C set/delete lower margin")

5.4.29 PROPORT/PROPORT_C – proportional spacing on/off

Synopsis

PROPORT:

ASCII:	ESC	p	1
Decimal:	27	112	49
Hexadecimal:	1B	70	31

PROPORT_C:

ASCII:	ESC	p	0
Decimal:	27	112	48
Hexadecimal:	1B	70	30

Description

PROPORT activates proportional spacing, e.g. the right and left spaces of a character are always of equal size. In normal type, each letter occupies the same position, regardless of its width. The proportional print is combined automatically with *LQ*, and the respective pitch information is canceled.

PROPORT_C deactivates proportional spacing.

Note

When using a text processing program which justifies both margins, the right margin only appears justified if the software can likewise handle proportional spacing.

5.4.30 RELPOS – set relative dot position

Synopsis

ASCII:	ESC	\	*n1*	*n2*
Decimal:	27	92	*n1*	*n2*
Hexadecimal:	1B	5C	*n1*	*n2*

Description

RELPOS places the print head at a dot position relative to the respective print position. The position is determined using the formula "*n2* * 256 + *n1*" and depends, unlike *ABSPOS*, on the respective character size and the letter quality selected (*DRAFT* or *LQ*). For *LQ*, a dot is 1/180 inches wide, and is 1/120 inches wide for *DRAFT*.

In order to determine *n1* and *n2*, the required displacement in dots must first be calculated. If the data should be moved to the left, the value of "65536" must be subtracted.

The values *n1* and *n2* can then be calculated using the following formula:

n1 = *n*MOD 256

n2 = INT(*n*/256)

Note

The sequence is ignored, if the controlled print position is outside the margins.

See also

ABSPOS (Section "ABSPOS set absolute dot position")

5.4.31 RESET – printer initialization

ASCII:	ESC	@
Decimal:	27	64
Hexadecimal:	1B	40

Synopsis

Description

RESET sets the printer to the default values and deletes from the print buffer all print data determined before the command, up to the last carriage return.

Note

In RS-232C printers, users can frequently set default values themselves.

5.4.32 SAN – switch to print station n

Synopsis

ASCII:	ES	[	=	<	3	5	;	*n*	;	s

	C									
Decimal:	27	91	61	60	51	53	59	*n*	59	115
Hexadecimal:	1B	5B	3D	3C	33	35	3B	*n*	3B	73

Description

SAN switches a printer to one of the following print stations:

nnn== 0:1 print station == form tractor 0/journal 1form
n == :2: 0print station == tractor 1/journal 2single sheet
3: 1print station feeding
2print station 3 (feeder)passbook/voucher

Note

If no print station number has been selected, the standard print station of the respective printer is used.

5.4.33 SI/SI_C – condensed print on/off

Synopsis

SI:

ASCII:	ESC	SI
Decimal:	27	15
Hexadecimal:	1B	0F

SI_C:

ASCII:	DC2
Decimal:	18
Hexadecimal:	12

Description

SI prints characters approx. 40% smaller than normal. Condensed print can neither be combined with bold type nor with 15-pitch character width.

SI_C deactivates condensed print activated with *SI*.

5.4.34 SLM – set left margin

Synopsis

ASCII:	ESC	|	*n*
Decimal:	27	108	*n*
Hexadecimal:	1B	6C	*n*

Description

SLM sets the left margin at *n* columns (0 <= *n* <= 160) in the respective character width. If, however, the margin exceeds 8 inches, the value is ignored. The absolute margin position depends on the character size and the print width. With proportional spacing, the left margin is determined according to the character size 10 pitch (Pica).

Note

SLM should be inserted at the start of a line, because it deletes all tabulators and characters entered previously in a print line. The spacing between the left and right margin should be at least one double-width Pica character.

See also

SLM_60 (Section "SLM_60 set left margin to 1/60 inches")

SRM (Section "SRM set right margin")

5.4.35 SLM_60 – set left margin to 1/60 inches

Synopsis

ASCII:	ESC	[	=	<	7	;	*n*	;	s
Decimal:	27	91	61	60	55	59	*n*	59	115
Hexadecimal:	1B	5B	3D	3C	37	3B	*n*	3B	73

Description

SLM_60 sets the left margin of a page/form in *n* units of 1/60 inches.

5.4.36 SLOW/SLOW_C – half speed on/off

Synopsis

SLOW:

ASCII:	ESC	s	1
Decimal:	27	115	49
Hexadecimal:	1B	73	31

SLOW_C:

ASCII:	ESC	s	0
Decimal:	27	115	48
Hexadecimal:	1B	73	30

Description

SLOW switches the printer to half speed. The noise level reached is consequently low.

SLOW_C switches the printer to normal speed.

5.4.37 S_TOP – set start of page

Synopsis

ASCII:	ESC	[	=	<	2	2	;	*n*	;	s
Decimal:	27	91	61	60	50	50	59	*n*	59	115
Hexadecimal:	1B	5B	3D	3C	32	32	3B	*n*	3B	73

Description

S_TOP sets the first printable line of a page/form in units of *n* = 1/24 inches. With a subsequent form feed (*FF*), this line is indented automatically. All subsequent line feeds are based on this start of page. If a line is specified which cannot be printed physically, the printer indents to the first printable line.

Note

SAS-II printers interpret "*n* = 0" as "*n* = 20".

5.4.38 SPM/SPM_C – shadow printing on/off

Synopsis

SPM:

ASCII:	ESC	G
Decimal:	27	71
Hexadecimal:	1B	47

SPM_C:

ASCII:	ESC	H
Decimal:	27	72
Hexadecimal:	1B	48

Description

SPM switches on shadow printing and can occur at any position within a print line. Every printable character is

generally printed twice (moved horizontally). *SPM* can extend beyond the line.

SPM_C switches off shadow printing and can occur at any position within a print line.

See also

BPM (Section "BPM/BPM_C bold type on/off")

5.4.39 SRM – set right margin

Synopsis

ASCII:	ESC	Q	*n*
Decimal:	27	81	*n*
Hexadecimal:	1B	51	*n*

Description

SRM sets the right margin at *n* columns (1 <= *n* <= 255) in the respective character width. The absolute margin position depends on the character size and the print width of the characters. If proportional spacing is on, the right margin is determined in accordance with the character size 10 pitch (Pica).

Note

SRM must be inserted at the start of a line, because it deletes all tabulators entered previously in a print line and all subsequent characters. The spacing between the left and right margin should be at least one double-width Pica character.

On reaching the right margin, a carriage return and a line feed code are always issued after this command.

See also

SLM (Section "SLM set left margin")

5.4.40 SS2/SS3 – set character table G2/G3

Synopsis

SS2:

ASCII:	non-displayable
Decimal:	142
Hexadecimal:	8E

SS3:

ASCII:	non-displayable
Decimal:	143
Hexadecimal:	8F

Description

The character following the *SS* sequence is generated from the G2 or G3 character table using the code range 20–7F.

Note

The sequence only affects the character directly following it. The previous settings then apply again.

The print control functions *LS*, *LSR* and *SS* can be mixed arbitrarily within a job.

The standard code table adopted is marked number "1" in the *iocs.dev* file.

See also

LS0, LS1, LS2, LS3 (Section "LS0/LS1/LS2/LS3 set character table G0/G1/G2/G3 to GL until next changeover")

LS1R, LS2R, LS3R (Section "LS1R/LS2R/LS3R set character table G1/G2/G3 to GR until next changeover")

SWCTAB (Section "SWCTAB select code table")

5.4.41 STYLE – select masterstyle

Synopsis

ASCII:	ESC	!	*n*
Decimal:	27	33	*n*
Hexadecimal:	1B	21	*n*

Description

STYLE is used to select every valid combination of the following print modes. The required mode is set with the number *n*, which can be determined by adding the values stated in the following table:

Print mode	Decimal	Hexadecimal
Pica (10 pitch) Elite (12 pitch) Proportional print Condensed print Bold type Double impact Wide print Italics Underlining	0 1 2 4 8 16 32 64 128	00 01 02 04 08 10 20 40 80

Note

NLQ font cannot be defined with the masterstyle command. Print quality and type font must be defined separately, and either with the font selection function or the commands *NLQ*, *LQ* and *FONT*. Moreover, if super- and subscript are set, they cannot be canceled.

The possible combinations vary from printer to printer.

See also

NLQ (Section "NLQ near letter quality on")

LQ (Section "LQ letter quality on")

FONT (Section "FONT select font")

5.4.42 SUBSCRIPT – subscript on

Synopsis

ASCII:	ESC	S	1
Decimal:	27	83	49
Hexadecimal:	1B	53	31

Description

SUBSCRIPT causes all subsequent characters to be printed half a line lower. *SUBSCRIPT* can be inserted at any point in a line and its effect extends beyond the current line.

Note

SUBSCRIPT is switched off with *SUBP_C*.

See also

SUPERSCRIPT (Section "SUPERSCRIPT superscript on")

SUBP_C (Section "SUBP_C super/subscript off")

5.4.43 SUPERSCRIPT – superscript on

Synopsis

ASCII:	ESC	S	0
Decimal:	27	83	48
Hexadecimal:	1B	53	30

Description

SUPERSCIRPT causes all subsequent characters to be printed half a line higher. *SUPERSCRIPT* can be inserted at any point in a line and its effect extends beyond the current line.

Note

SUPERSCRIPT is switched off with *SUBP_C*.

See also

SUBSCRIPT (Section "SUBSCRIPT subscript on")

SUBP_C (Section "SUBP_C super/subscript off")

5.4.44 SUBP_C – super/subscript off

Synopsis

ASCII:	ESC	T
Decimal:	27	84
Hexadecimal:	1B	54

Description

SUBP_C switches off *SUBSCRIPT* and *SUPERSCRIPT*. Depending on the preceding function, the paper is moved backwards or forwards by a half a line.

See also

SUBSCRIPT (Section "SUBSCRIPT subscript on")

SUPERSCRIPT (Section "SUPERSCRIPT superscript on")

5.4.45 SWCCC – select control character class

Synopsis

ASCII:	ESC	[	=	<	9	9	;	*n1*	;	*n2*	s
Decimal:	27	91	61	60	57	57	59	*n1*	59	*n2*	115
Hexadecimal:	1B	5B	3D	3C	39	39	3B	*n1*	3B	*n2*	73

Description

SWCCC specifies for a job the control character classes to which the transferred control characters should be assigned.

The following control character classes are defined at present:

PR_NATIVE Data are not analysed
PR_EPSON Data have EPSON ESCs (default)
PR_DPTG Data in accordance with DPTG protocol

The control character class *PR_EPSON* requires the EPSON sequences and data in ISO 8-bit format as input code. It is possible to switch to *PR_NATIVE* and *PR_DPTG* for a certain number of characters, so that special jobs can also be supported within a job (refer to the Section "Outputting a data string" in the Chapter "User interface for printers"):

n1 = 1: Switch to *PR_NATIVE*

n1 = 2: Switch to *PR_DPTG*

n2 = Amount of subsequent data in ASCII representation (max. 3 position)

5.4.46 SWCTAB – select code table

Synopsis

ASCII:	ESC	[	=	<	9	8	;	*n*	;	s
Decimal:	27	91	61	60	57	56	59	*n*	59	115
Hexadecimal:	1B	5B	3D	3C	39	38	3B	*n*	3B	73

Description

SWCTAB allows any code table to be set within a job with control character class *PR_EPSON*, or allows the code conversion to be switched off:

n = 0

The code conversion is switched off.

n > 0

The corresponding code table is set. The assignment between the specified table number and the corresponding code table is defined in the file *iocs.dev*. The table number is specified in a maximum of 3 position ASCII representation.

See also

LS0, LS1, LS2, LS3 (Section "LS0/LS1/LS2/LS3 set character table G0/G1/G2/G3 to GL until next changeover")

LS1R, LS2R, LS3R (Section "LS1R/LS2R/LS3R set character table G1/G2/G3 to GR until next changeover")

5.4.47 UL/UL_C – underlining on/off

Synopsis

UL:

ASCII:	ESC	_	1
Decimal:	27	45	49
Hexadecimal:	1B	2D	31

UL_C:

ASCII:	ESC	_	0
Decimal:	27	45	48
Hexadecimal:	1B	2D	30

Description

UL switches on underlining on the printer and can occur at any position within a print line. Every printable character, including blanks, after this position will be underlined. *UL* can extend beyond the line.

UL_C switches off underlining and can occur at any position within a print line.

5.4.48 UNIDIR/UNIDIR_C – unidirectional print on/off

Synopsis

UNIDIR:

ASCII:	ESC	U	1

UNIDIR_C

ASCII:	ESC	U	0

Decimal:	27	85	49
Hexadecimal:	1B	55	31

Decimal:	27	85	48
Hexadecimal:	1B	55	30

Description

UNIDIR activates unidirectional print, to ensure precise positioning of characters when printing text.

UNIDIR_C switches from unidirectional printing to bidirectional printing.

Note

Normally, printing is bidirectional.

5.4.49 VT – vertical tabulation

Synopsis

ASCII:	VT
Decimal:	11
Hexadecimal:	0B

Description

VT forces a paper feed to the next vertical tabulator after printing the current line. The vertical tabulators are set with *VT_SET*. If no vertical tabulator is set, there is a line feed by one line.

See also

VT_SET (Section "VT_SET set vertical tabulator at line n")

5.4.50 VT_SET – set vertical tabulator at line n

Synopsis

ASCII:	ESC	B	*n1*	*n2*	...	*n16*	NUL
Decimal:	27	66	*n1*	*n2*	...	*n16*	0
Hexadecimal:	1B	42	*n1*	*n2*	...	*n16*	00

Description

V*T_SET* sets 1 to 16 vertical tabulators. These are defined by *n1*–*n16*, whereby the values lie between "1" and "255" and must be in ascending order. The line numbers are absolute values, i.e. they always refer to the start of the page. It is not permitted to extend beyond the page.

See also

VT (Section "VT vertical tabulation")

5.5 Control sequences of the individual printers

The tables below list the control sequences supported by individual printers. The meanings of the abbreviations used are as follows:

X = Sequence is executed by the printer.

X1 = Sequence is executed depending on the font cassettes used (HP Laserjet).

X2 = Sequence is switched off for line feeds (Diablo 630).

X3 = Double width and double height cannot be switched on simultaneously; *DCHH_C* and *EPM_C* switch off both double height and double width.

_ = Sequence is ignored.

Abbreviation	**Printer name (short form)**				
	4007	**4009**	**4010**	**4014**	**4806**
ABSPOS	X	X	X	X	–
AFEN(_C)	–	–	–	X	–
BELL	X	–	–	X	–
BMARGIN(_C)	X	X	X	X	–
BPM(_C)	X	X	X	X	X
BS	X	X	X	X	X
COLOUR	–	–	–	X	–
CPI_10	X	X	X	X	X
CPI_12	X	X	X	X	X
CPI_15	X	–	X	X	X
CR	X	X	X	X	X
CURSIV(_C)	X	X	X	X	X
DCHH(_C)	X	–	X	–	–
DRAFT	X	X	X	X	–
EPM(_C)	X	X	X	X	–
FEEDER1/2	X	X	X	X	–
FEJ	–	X	X	X	X
FF	X	X	X	X	X
FONT	X	–	X	X	–
HT	X	X	X	X	–
HT_SET	X	X	X	X	–
LF	X	X	X	X	X
LFB	–	–	X	X	X
LPI_3/5/60	X	–	X	X	–
LPI_6	X	X	X	X	X
LPI_8	X	X	X	X	X
LQ	X	X	X	X	–
LS0/1/2/3	X	X	X	X	X
LS1R/2R/3R	X	X	X	X	X
NLQ	X	X	X	X	–
PLENGTH	X	X	X	X	–

PROPORT(_C)	X	X	X	X	X
RELPOS	X	X	X	X	–
RESET	X	X	X	X	X
SAN	–	–	–	–	–
SI(_C)	X	X	X	X	X
SLM	X	X	X	X	X
SLM_60	–	–	–	–	–
SLOW(_C)	X	X	X	–	–
S_TOP	–	–	–	–	–
SPM(_C)	X	X	X	X	X
SRM	X	X	X	X	X
SS2/SS3	X	X	X	X	X
STYLE	X	X	X	X	–
SUBSCRIPT	X	X	X	X	–
SUPERSCRIPT	X	X	X	X	–
SUBP_C	X	X	X	X	–
SWCCC	X	X	X	X	X
SWCTAB	X	X	X	X	X
UL(_C)	X	X	X	X	X
UNIDIR(_C)	X	X	X	X	–
VT	X	X	X	X	–
VT_SET	X	X	X	X	–

Abbreviation	**Printer name (short form)**									
	4810	**4815**	**4904**	**9014**	**9021**	**9022 HPLJ**	**9022 D630**	**9041/42**	**9043/44**	**MD06**
ABSPOS	–	–	X	X	–	–	–	–	–	–
AFEN(_C)	–	–	–	–	–	–	–	–	–	–
BELL	–	–	–	–	–	–	–	–	X	X
BMARGIN(_C)	–	–	X	X	–	–	–	–	–	X
BPM(_C)	X	X	X	X	X	X1	X2	–	–	X
BS	X	X	X	X	X	X	X	X	X	X
COLOUR	–	–	–	–	–	–	–	–	–	–
CPI_10/_12	X	X	X	X	X	X	X	–	–	X
CPI_15	X	X	X	X	X	X	X	–	–	–
CR	X	X	X	X	X	X	X	X	X	X
CURSIV(_C)	X	X	X	X	X	X1	–	–	–	X
DCHH(_C)	–	–	–	X	–	–	X3	–	–	X

DRAFT/LQ/NLQ	–	–	X	X	–	–	–	–	–	X
EPM(_C)	–	–	X	X	–	–	X3	–	–	X
FEEDER1/2	X	X	–	X	–	X	X	–	–	X
FEJ	X	X	X	X	–	X	X	–	–	–
FF	X	X	X	X	X	X	X	X	X	X
FONT	–	–	X	X	–	–	–	–	–	X
HT	X	–	X	X	–	–	X	X	X	X
HT_SET	–	–	X	X	–	–	–	–	–	X
LF	X	X	X	X	X	X	X	X	X	X
LF_LINES	X	X	X	X	X	X	X	X	X	X
LFB	X	X	X	–	X	X	X	–	–	–
LPI_3/_5/_60	–	–	X	X	–	–	–	–	–	X
LPI_6/_8	X	X	X	X	X	X	X	–	–	X
LS0/1/2/3	X	X	X	X	X	X	X	X	X	X
LS1R/2R/3R	X	X	X	X	X	X	X	X	X	X
PLENGTH	–	–	X	X	–	–	X	–	–	X
PROPORT(_C)	X	X	X	X	X	X1	X	–	–	X
RELPOS	–	–	X	X	–	–	–	–	–	–
RESET	X	X	X	X	X	X	X	–	X	X
SAN	–	–	X	–	–	–	–	–	–	–
SI(_C)	X	X	X	X	–	X	–	–	–	X
SLM	X	X	X	X	X	X	X	–	–	X
SLM_60	–	–	–	–	–	–	–	–	–	–
SLOW(_C)	–	–	–	–	–	–	–	–	–	–
S_TOP	–	–	X	–	–	–	–	–	–	–
SPM(_C)	X	X	X	X	–	X1	X2	–	X	X
SRM	X	X	X	X	X	X	–	–	–	X
SS2/SS3	X	X	X	X	X	X	X	X	X	X
STYLE	–	–	X	X	–	X1	X	–	–	–
SUBSCRIPT	–	–	X	X	–	–	X	–	–	X
SUPERSCRIPT	–	–	X	X	–	–	X	–	–	X
SUBP_C	–	–	X	X	–	–	X	–	–	X
SWCCC	X	X	X	X	X	X	X	X	X	X
SWCTAB	X	X	X	X	X	X	X	X	X	X
UL(_C)	X	X	X	X	X	X	X	–	–	X
UNIDIR(_C)	–	–	X	X	–	–	–	–	–	X

VT	–	–	X	X	–	–	X	X	X	X
VT_SET	–	–	X	X	–	–	–	–	–	X

Abbreviation	Printer name (short form)					
	MD14	ND24/25/27/44/45, TD06/08		ND31/33	ND37	ZD09
		Leporello	Feeder			
ABSPOS	–	X	X	–	X	–
AFEN(_C)	–	–	X	–	–	–
BELL	–	–	–	X	X	–
BMARGIN(_C)	–	X	X	X	X	–
BPM(_C)	X	X	X	X	X	–
BS	X	X	X	X	X	–
COLOUR	–	–	–	–	–	–
CPI_10/_12	X	X	X	X	X	–
CPI_15	X	X	X	–	X	–
CR	X	X	X	X	X	X
CURSIV(_C)	X	–	–	–	–	–
DCHH(_C)	–	–	–	–	–	–
DRAFT/LQ/NLQ	–	X	X	X	X	–
EPM(_C)	–	X	X	X	X	–
FEEDER1/2	X	–	X	–	–	–
FEJ	X	–	X	–	X	–
FF	X	X	X	X	X	X
FONT	–	–	–	–	–	–
HT	–	X	X	X	X	X
HT_SET	–	X	X	X	X	–
LF	X	X	X	X	X	X
LF_LINES	X	X	X	X	X	X
LFB	X	–	–	X	–	–
LPI_3/_5/_60	–	X	X	–	X	–
LPI_6/_8	X	X	X	X	X	–
LS0/1/2/3	X	X	X	X	X	X
LS1R/2R/3R	X	X	X	X	X	X
PLENGTH	–	X	X	X	X	–
PROPORT(_C)	X	–	–	X	X	–
RELPOS	–	–	–	–	X	–
RESET	X	X	X	X	X	–

SAN	–	X	X	–	–	–
SI(_C)	X	X	X	X	X	–
SLM	X	X	X	X	X	–
SLM_60	–	X	X	–	–	–
SLOW(_C)	–	–	–	X	X	–
S_TOP	–	X	X	–	–	–
SPM(_C)	X	X	X	X	X	–
SRM	X	X	X	X	X	–
SS2/SS3	X	X	X	X	X	X
STYLE	–	X	X	X	X	–
SUBSCRIPT	–	X	X	X	X	–
SUPERSCRIPT	–	X	X	X	X	–
SUBP_C	–	X	X	X	X	–
SWCCC	X	X	X	X	X	X
SWCTAB	X	X	X	X	X	X
UL(_C)	X	X	X	X	X	–
UNDIR(_C)	–	X	X	X	X	–
VT	–	X	X	X	X	X
VT_SET	–	X	X	X	X	–

6 Installation of IOCS

6.1 Preface

IOCS is installed with the command *pkgadd(1M)*, and comprises the following components:

Package	Content
SIiocsR	IOCS runtime system
SIiocsT	IOCS table tool
SIiocsM	IOCS message package
SIiocsD	IOCS development system (optional)

If you are installing the packages individually instead of using the *all* option, the *SIiocsR* package must be installed first. Only then can the other packages be installed.

It is only necessary to install the development system if an application is to be linked to the *libxio.a* library, if the IOCS table tool is to be used, or if a separate device control process is to be generated. The IOCS table tool and the generation of separate device control processes are described in the manual "Generating Device Control Processes".

IOCS V3.0 is downward compatible with IOCS V2.4; it requires SINIX V5.41 and the logging package *SIlog3*. The *SIxmsb* package (XMS runtime system) must be installed for evaluating the log entries and for the IOCS development system. If the IOCS table tool is invoked with *pas*, the *SImenur* package must be installed.

6.1.1 Purpose

This section of the manual describes the installation and deinstallation of IOCS.

6.1.2 Layout of section

The Section "First installation" describes a first-time installation of IOCS on your system.

The Section "Reinstallation" tells you how to update an existing installation of IOCS on your system.

The Section "Deinstallation" describes how to deinstall IOCS.

6.2 First installation

When installing the IOCS runtime system for the first time, the name of a base directory (where the runtime system is to be installed) is requested in dialog mode. The default directory */opt/iocs* can also be selected by simply hitting <CR>. The path name of the base directory is written to the *$IOCS* variable during installation.

The base directory specified is then automatically used when subsequently installing the IOCS table tool and the IOCS message package.

The first time the IOCS development system is installed, the specified base directory is not taken automatically; you are requested to specify the name of a directory where the IOCS development system is to be installed. The base directory of the installed IOCS runtime system can also be selected by hitting <CR>. The path name of this directory is written to the *$IOCSD* variable during installation.

When the IOCS runtime system is installed for the first time, the necessary configuration files *iocs.dev*, *iocs.fmod* and *iocs.tty* are created, and the required entries are automatically made in the */etc/profile* file. When the installation process is complete, you must execute the */etc/profile* file in order to set up the correct environment.

The response file *response* is also created (see the *-r* option under *pkgadd(1M)*). This file contains the values entered interactively during the installation, so that they can be used for follow-up installations, for example remote installations on other systems. If the *-r* option is specified for a follow-up installation, the stored values are accessed and no information is requested interactively.

After the installation process has been concluded successfully, IOCS must be configured (see the Chapter

"Configuration of IOCS"). IOCS is then ready for operation.

When installing the packages *SIiocsR*, *SIiocsT*, and *SIiocsM*, the following directories and files are created:

Name	Comment
$IOCS/bin	This directory contains all the necessary programs.
$IOCS/fifos	This directory contains the FIFOs at IOCS runtime; in the case of applications with the library from IOCS V2.4, their process trace files are contained here.
$IOCS/include	This directory contains the *iocs_config.h* description file.
$IOCS/iocs	For compatability reasons, this directory is linked to the *$IOCS/tables* directory.
$IOCS/lib	This directory contains the configuration files *iocs.dev*, *iocs.fmod*, and *iocs.tty*.
$IOCS/misc	This directory contains the response file for subsequent pakkage installations, and the profile entries for IOCS.
$IOCS/tables	This directory contains the code and emulation tables for the table tool.
$IOCS/tmp	For compatability reasons, this directory is linked to the $IOCS/fifos directory.
*$IOCS/xm*s	The directory contains the logging message files.
/usr/lib	This directory contains the application libraries *libxio.so* and *libxio_t.so*, as well as the libraries *libgsp.so* and *libgsp_t.so* for the device control processes.
/etc/Iocs	This file contains the installation path.

When installing the *SIiocsD* package, the following directories and files are created:

Name	Comment
$IOCSD/bin	This directory contains the *iocstool* program.
$IOCSD/lib	This directory contains the static application libraries *libxio.a* and *libxio_t.a*, as well as the static libraries *libgsp.a* and *libgsp_t.a* for the device control processes.
$IOCSD/misc	This directory contains profile entries for *IOCSD*, and an example for programming your own device control process.
/usr/include	This directory contains the header files *xio.h*, *xio_desc.h*, and *xio_gsp.h*.
/etc/Iocsd	This file contains the installation path.

6.3 Reinstallation

Before reinstalling IOCS V3.0, all current IOCS applications and the active central process must be terminated.

If IOCS V2.4 is already installed on the system, the *SIiocs* package must be deleted with the *pkgrm(1M)* command. The process is then continued as for a first installation.

If you wish to change the base directory of an IOCS V3.0 system already installed, all packages must be deleted with the *pkgrm(1M)* command in the following order: *SIiocsT*, *SIiocsM*, *SIiocsD*, *SIiocsR*. The process is

then continued as for a first installation.

If you wish to reinstall IOCS V3.0 in the same base directory as it is already installed, the name of the base directory is not requested during the installation process. Existing configuration files are not overwritten. Otherwise, the process is the same as for a first installation.

6.4 Deinstallation

Before deinstalling IOCS V3.0, all current IOCS applications and the active central process must be terminated. The packages must then be deleted with the *pkgrm(1M)* command in the following order: *SIiocsT*, *SIiocsM*, *SIiocsD*, *SIiocsR*.