English

openFT V12.0 for Unix Systems and Windows Systems

openFT-Script Interface

User Guide

Edition September 2012

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to:
manuals@ts.fujitsu.com

Certified documentation according to DIN EN ISO 9001:2008

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2008.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright and Trademarks

Contents

1 Preface

This document describes Version 1 of openFT-Script.

openFT-Script is a language for the description of multiple logically interdependent openFT requests. openFT-Script makes it possible to combine these requests to form a single request (Ftscript).

openFT-Script reduces the customer workload involved in monitoring sequential openFT requests and permits restarts in the event of a downtime.

openFT-Script uses XML notation.

You can use openFT-Script under Windows™ as well as under Unix systems.

1.1 Target group

This manual is intended for XML programmers who want to create openFT-Script requests. openFT-Script requests are used to start openFT requests in Windows or Unix systems and, for example, transfer files to or from other systems.

A knowledge of Windows and Unix-based operating systems as well as an understanding of XML would be useful when reading this manual.

The manual applies to Windows systems, Solaris systems and to portings to other Unix platforms. The operating system-dependent differences are described in detail in the Release Notice which is supplied with each product CD.

1.2 Concept of openFT manuals for Unix and Windows systems

The complete description of openFT and its components comprises a number of different manuals. Alongside the present manual there are also other openFT manuals for Unix systems and Windows systems.
This description is distributed across the manuals as follows:

- openFT for Unix systems - Installation and Administration

  The system administrator manual is intended for FT, FTAC and ADM administrators. It describes:
  - the installation of openFT and its optional components
  - the operation, control and monitoring of the FT system and the FTAC environment
  - the administration commands for FT and FTAC administrators
  - the configuration and operation of a remote administration server and a ADM trap server
  - important CMX commands.

- openFT for Windows systems - Installation and Administration

  The system administrator manual is intended for FT, FTAC and ADM administrators. It describes:
  - the installation of openFT and its optional components
  - the operation, control and monitoring of the FT system and the FTAC environment
  - the administration commands for FT and FTAC administrators
  - the configuration and operation of a remote administration server and a ADM trap server

- openFT for Unix systems - Managed File Transfer in the Open World

  The user manual is intended for the openFT user and describes:
  - the basic functions of the openFT product family,
  - the conventions for file transfers to computers running different operating systems,
  - details on implementing FTAM,
  - the openFT user commands,
  - the openFT-Script commands,
  - the BSFT interface,
  - the messages of the different components.

- openFT Windows systems - Managed File Transfer in the Open World

  The user manual is intended for the openFT user and describes:
  - the basic functions of the openFT product family,
  - the conventions for file transfers to computers running different operating systems,
  - details on implementing FTAM,
  - the openFT user commands,
  - the openFT-Script commands,
  - the messages of the different components.

- openFT for Unix systems and Windows systems - Program Interface

  This manual is intended for C programmers and describes the C program interface on Unix and Windows systems.

- openFT for Unix systems and Windows systems - openFT-Script Interface

  This manual is intended for XML programmers and describes:
  - the openFT-Script commands
  - the XML statements for the openFT-Script interface

i Many of the functions described in the manuals are also available in the openFT graphical interface (open FT Explorer). A detailed online help system that describes the operation of all the dialogs is supplied together with the openFT Explorer. The online help system also contains a complete description of the openFT commands.

1.3 Changes compared to the predecessor version

The openFT-Script interface to openFT V12 provides the following new commands for the variable issue of openFT-Script requests:

- *ftmodsuo* for modifying openFT-Script user options.
- *ftshwsuo* for displaying openFT-Script user options.

The commands *ftcans*, *ftdels* and *ftshws* have been modified:

- *ftscriptid:* You can use the wildcard characters ? and * in the *ftscriptid* in order to identify the openFT-Script request.

The amount of memory that can be allocated in the Java Virtual Machine in order to execute Ftscripts has been increased in openFT V10.0B20, with the result that even extensive openFT-Script requests with high memory requirements can be run. At the same time, resource consumption has been regulated through the use of parallel threads (see section “parallel” on page 82).

1.4 Notational conventions

The following notational conventions are used throughout this manual:

`typewriter font`
: typewriter font is used to identify entries and examples.

italics
: In running text, commands, statements, names, variables and values are indicated by italic letters, e.g. file names and host names.

i indicates notes

Additional conventions are used for the command descriptions and the program interface.

1.5 README files

Information on any functional changes and additions to the current product version can be found in product-specific README files.

1.6 Requirements for openFT-Script

openFT-Script is supplied with openFT and requires an openFT version as of V10 to be installed on the executing host. All the addressed partners must use an FTAM-/openFT-compatible product for file transfer.

If openFT is not used then the restrictions described in the openFT manual apply.

At least J2SE^{TM} Runtime Environment 5.0 (JRE 5.0) is required for the Java runtime environment. On Windows systems, the extended language version of Java JRE (support of non-European languages, Extended Encoding Set) is also required. This does not have to be explicitly installed if Java JDK or Java 1.6 (or higher) is installed.

2 Structure of an Ftscript

2.1 Components of an Ftscript

An Ftscript consists of activities. Each activity has a context. The context may also describe error handling mechanisms (*faulthandler*).

- **Activities** may take the form of instructions issued to openFT (e.g. *transferFile*, *deleteFile*) or instructions which control the workflow (e.g. *parallel*, *foreach*). The instructions are described in chapter “openFT-Script statements” on page 51.
- Files, directories, scripts and partners can be stored as context objects in the **context**. By means of **referencing**, it is possible to re-use this type of context object in the activity and the underlying activities provided that these do not possess a context object with the same name. Re-use is not possible outside of the activity in which the context object is defined.
- If a fault occurs within an activity then **error handling** can be used to supply an appropriate response. If error handling terminates correctly (i.e. without errors) then the associated activity is considered to have been completed successfully. Similarly, if error handling is terminated with an error then the activity is considered to be defective.

The general structure of an Ftscript is as follows:

```
<ftscript version="1">
  <context>
    ContextObjects*
    faulthandler?
  </context>
  Activities+
</ftscript>
```

Every Ftscript has the statement *<ftscript version="1">* as its root element.
The root element contains the following sub-elements:

– an (optional) context with context objects and a maximum of one *faulthandler*
– one or more activities which are executed in the specified sequence

For further information on the syntax, see section “Syntax of the openFT-Script statements” on page 51.

2.1.1 Activities

There are various types of activity

Internal activities

Internal activities consist of instructions sent to the Ftscript interpreter to control operation.

These include *ftscript*, *sequence*, *parallel*, *foreach*, *empty* and *fault*
(For a description, see chapter “openFT-Script statements” on page 51).

External activities

The external activities are statements issued to openFT instructing it to run the required functions.

These include *executeScript*, *transferFile*, *deleteFile*, *createDirectory*, *deleteDirectory* and *listDirectory*
(For a description, see chapter “openFT-Script statements” on page 51).

If you do not specify a partner in the external statements *executeScript*, *deleteFile*, *createDirectory*, *deleteDirectory* or *listDirectory* then the statement is executed as a local command or local operating system statement.

Parent and child activities

You can nest activities (XML syntax). As a result, activities are subdivided into parent and child activities.

```
<ftscript version="1">
  <parallel>
    <transferFile …/>
    <listDirectory …/>
    <foreach …>
      <deleteFile …/>
    </foreach>
  </parallel>
</ftscript>
```

ftscript is the root element.

The root element has a child element (*parallel*).
parallel has *ftscript* as parent element (or higher-level element).

parallel also has three child elements (*transferFile*, *listDirectory* and *foreach*). The *foreach* activity also has a *deleteFile* activity as a child element.

2.1.2 Context

An activity's context describes the context objects and error handling mechanisms (*faulthandler*). Using the "context object" language tool, you can specify an element, for example a partner, once in the Ftscript and then re-use it whenever necessary. To do this, you reference the context object at the point at which it is to be used. Using appropriate referencing, it is also possible to combine the properties of multiple context objects.

Each context object has an ID which must be unique within the context. This ID is used to address (reference) the context object.

If a referenced context object is not found in the current context then a (recursive) search is performed in the higher-level contexts. Context objects other context objects with the same ID in higher-level contexts.

A context is always present even if it has not been defined. There are certain activities (*foreach*, *listDirectory*) which automatically incorporate data in a context.

The context objects are described in more detail in section “context” on page 54.

2.1.3 Referencing

openFT-Script also allows you to combine context objects.
For example, a partner (see section “partner” on page 84) does not have to be specified in full at every location. You can transfer a partner to a context object and re-use it via referencing.

The following rules apply to de-referencing:

1. A *ref* attribute references a context object with the specified ID and the object type of the context object (e.g. file, partner).
2. The search is continued in the parent context if no suitable object is found in the current object. If no suitable context object is found there then the search is continued in its parent context.
   Multi-level referencing is permitted.
3. If no suitable context object is found then the script is terminated with the error *ft_reference* (see section “Error handling” on page 22).
4. *ref="A"* is permitted in a context object with *ID="A"*. In this case,
   ref always refers to the parent context.
5. Circular references are nor permitted:

   obj1(ID="A", ref="B"); *obj2(ID="B", ref="A")* is not possible in one and the same context. The error *ft_reference* is output.
6. All the attributes and elements of the referenced element which are not present in the referenced element are taken over. Before being taken over, the referenced element is de-referenced on the basis of these rules.

Examples

1. Valid referencing

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <partner id="remote" name="WindowsP_1">
      <transferAdmission>
        <ftacAdmission ftacAdmission="FTACADM"/>
      </transferAdmission>
    </partner>
    <file id="pack" name="pack1.bin">
      <partner ref="remote"/>
      <directory name="frg_eis_01"/>
    </file>
  </context>
  <transferFile>
    <fromRemoteFile ref="pack"/>
    <toLocalFile name="pack1.bin">
      <directory name="frg_eis_01"/>
    </toLocalFile>
  </transferFile>
  <transferFile>
    <context>
      <partner id="remote" name="UnixP_1">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM"/>
        </transferAdmission>
      </partner>
    </context>
    <fromRemoteFile ref="pack">
      <partner ref="remote"/>
    </fromRemoteFile>
    <toLocalFile name="pack2.bin">
      <directory name="frg_eis_01"/>
    </toLocalFile>
  </transferFile>
</ftscript>
```

The file object with the *ID="pack"* references a partner object with *ID="remote"*.
The partner *WindowsP_1* is used in the first *transferFile* activity.

The same file object *pack* is referenced in the second *transferFile* activity. However, the partner has been overwritten. Consequently, the partner *UnixP_1*, which is defined in the context of the second *transferFile*, is addressed. This definition hides the Windows partner with *Id="remote"*.

If the partner was not overwritten then the Windows partner would be used since this is found in the *pack* object definition in the *ftscript* context. The partner (on the Unix system) from the context of the second *transferFile* would be ignored since it is no longer accessible from the definition *Id="pack"*.

2. Invalid referencing

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <transferFile>
    <fromRemoteFile ref="pack"/>
    <toLocalFile name="pack1.bin">
      <directory name="frg_eis_02"/>
    </toLocalFile>
  </transferFile>
</ftscript>
```

The *pack* reference is not defined. The script outputs an error *ft_noRef*. This is not processed (no *faulthandler* defined). The script is terminated before execution of the *transferFile* activity. No restart is possible. You can use *ftshws* to display the reason for termination.

2.2 Specifying file and directory names

Many openFT-Script statements use file or directory names. These are specified in the attributes *name*, *bs2000Name*, *unixName*, *windowsName* and *zosName*. If no special characteristics are defined then the following definition applies.

If a partner is specified then its operating system must also be explicitly specified. openFT-Script itself determines the operating system of the local computer.

Rules for file name and directory name attributes

- An operating system-specific file name or directory name attribute (*bs2000Name*, *unixName*, *windowsName* oder *zosName*) is only evaluated, if it does not contain an empty string and if the specified or determined operating system has this type.
- If no matching operating system-specific file name attribute is found or it is specified as an empty string then the non-operating system-specific file name attribute *name* applies. Please note that the default value for *name* is the empty string.
- If you use a profile in which a file name or a directory name is defined as the transfer admission (see the openFT User Manual), then the easiest way is to omit all file name or directory name attributes. This way the empty string is used (= default value of the non-operating system-specific attribute, see above).

  Alternative: Enter an empty string for all relevant operating system-specific file name or directory name attributes. You can then omit the non-operating system-specific attribute or specify it as an empty string

2.2.1 File name attributes

If a file name starts with the pipe character ("|") then this has the special meaning of a generating (*fromLocalFile*, *fromRemoteFile*) or receiving (*toLocalFile*, *toRemoteFile*) program call (see the openFT user manual).

Restriction: File names can be a maximum of 512 characters long. This restriction is checked at the time the openFT-Script request is issued (static check). Depending on the operating system, the number of permitted characters may be less (see the openFT User Manual). This operating system-specific length is not checked until the Ftscript is run (dynamic check).

Please note the section "Rules for file name and directory name attributes" on page 19.

Name	**Value**	**Meaning**
`name?`	string | <u>" "</u>	File name with specification of the subpath (see openFT user manual). *name* is used if no operating system is known or no operating system-specific name has been specified. The default value is an empty string. If *name* is not specified and the profile *partner-ftac* is linked to a fixed file name then this applies. Only the character "/" may be used as a separator (e.g. */C:/x/y* in Unix systems or *C:/x/y* in Windows systems).
`bs2000Name?`	string | " "	File name for BS2000. The name is used if it has been specified and the associated partner is a BS2000 system.
`window-sName?`	string | " "	File name for Windows. The name is used if it has been specified and the associated partner or the associated local system is a Windows system. You can use the "/" notation or the Windows-specific notation.
`unixName?`	string | " "	File name for Unix system. The name is used if it has been specified and the associated partner or the associated local system is a Unix system. You may only use the "/" notation.
`zosName?`	string	File name for z/OS. The name is used if it has been specified and the associated partner is a z/OS system.

2.2.2 Directory name attributes

Restriction: Directory names can be a maximum of 512 characters long. This restriction is checked at the time the openFT-Script request is issued (static check). Depending on the operating system, the number of permitted characters may be less (see the openFT User Manual). This operating system-specific length is not checked until the Ftscript is run (dynamic check).

Name	**Value**	**Meaning**
`name?`	string | ""	Directory name. *name* is used if no operating system is known or no operating system-specific name has been specified. The default value is an empty string. If *name* is not specified and the profile *partner-ftac* is linked to a fixed directory name then this applies. Only the character "/" may be used as a separator.
`bs2000Name?`	string	BS2000-specific addressing (see openFT User Guide).
`window-sName?`	string	Windows-specific path. The name is used if it has been specified and the associated partner or the associated local system is a Windows system. You can use the "/" notation or the Windows-specific notation.
`unixName?`	string	Unix system specific path. The name is used if it has been specified and the associated partner or the associated local system is a Unix system. You may only use the "/" notation.
`zosName?`	string	z/OS-specific addressing (see openFT User Guide).

2.3 Error handling

The Ftscript is checked when it is read.
If errors are identified then the Ftscript is not run. When the Ftscript is started, you see a corresponding error message and a return code (see section “ftscript - Starting an openFT-Script request” on page 38).

If the check is completed successfully then the Ftscript is executed asynchronously.
If an error occurs during execution then the cause of the error is logged.

Every error message has a unique internal code and is assigned to a specific error code. The errors are assigned to one of the following two categories on the basis of the error codes:

– "normal" Ftscript error codes

– "severe" Ftscript error codes

The table listing all these error messages can be found in chapter “Error messages” on page 115.

The error codes assigned by openFT-Script always start with "ft_".
You can also use the *fault* activity (see section “fault” on page 65) to assign any other error codes. However, these may not begin with "ft_". These are always considered to be "normal" error codes.

The cause of the error can be displayed with *ftshwact*.

If no *faulthandler* exists for an error then *ftscript* is terminated with an error (status F). The status and cause of termination can be displayed using *ftshws*.

2.3.1 "Normal" Ftscript error codes

"Normal" errors relate to the objects which are to be transferred or to the involved computer.

A "normal" error can be intercepted by the *faulthandler (default)* (see section “default” on page 71).

error codes	**Description**
`ft_access`	It is not possible to access a file/directory/computer.
`ft_admin`	Administration error
`ft_auth`	Authentication error (incorrect ID/password/authorizations).
`ft_cantCreate`	It is not possible to create a file/directory.
`ft_cantDelete`	It is not possible to delete a file/directory.
`ft_configuration`	Configuration error.
`ft_connection`	A connection error has occurred.
`ft_corrupt`	A file does not correspond to the expected format.
`ft_exist`	A file/directory already exists.
`ft_localFileStructure`	Error in the local file.
`ft_notEmpty`	A directory for deletion is not empty.
`ft_notExist`	A file/directory/CCS/file owner does not exist.
`ft_remoteFileStructure`	Error in the remote file.

2.3.2 "Severe" Ftscript error codes

The "severe" errors are primarily caused by internal problems (e.g. insufficient storage space on the hard disk for administrative information) or script errors (e.g. unresolved references).

If a "severe" error occurs than a restart is only possible under certain conditions (see section “Restart” on page 25).

"Severe" errors cannot be intercepted by the *faulthandler (default)*. In this case, it is necessary to write an explicit *faulthandler (case)* (see section “case” on page 70).

Error codes	**Description**
`ft_abort`	Termination by the user.
`ft_error`	A general error has occurred.
`ft_notSupported`	The functionality is not supported.
`ft_panic`	A serious error has occurred.
`ft_paramError`	A parameter error has occurred.
`ft_paramTooLong`	A parameter lies outside the value range.
`ft_recoveryFailed`	Error during restart (see section “Restart” on page 25). It is not possible to intercept the error.
`ft_reference`	A reference is invalid (not present or circular).
`ft_resource`	A resource error has occurred. (e.g. not enough storage space)
`ft_syntax`	A syntax error has occurred.

2.3.3 Restart

Ftscripts can generally be restarted if, for example, they have been aborted due to a system crash. Restrictions apply only to the following activities:

- *executeScript*, if *repeatable=no* was specified
- *createDirectory*, if *faultIfExists* was specified
- *deleteFile* or *deleteDirectory* if *faultIfNotExists* was specified

If the openFT-Script request is aborted during the processing of the statement then it is not possible to determine whether the activity has been completed. In the above cases it is not clear, when the restart is performed, how the Ftscript should continue to run.

If, for example, a directory that was to be created already exists then it is not possible to determine whether it was created by the aborted *createDirectory* activity or whether it already existed before the openFT-Script request was run.

If the restart operation encounters this type of ambiguous situation then it reacts as follows:

Activity	**Response on restart**
`executeScript` with the attribute `repeatable=no`	Activity aborted with the error *ft_resumeForbidden*
`createDirectory` with the attribute `faultIfExists`	If the directory already exists then activity aborted with the error *ft_recoveryCreateDirectory*
`deleteFile` or `deleteDirectory` with the attribute `faultIfNotExists`	If the file does not (or no longer?) exist(s) then the activity is aborted with the error *ft_recoveryFailed*

This response may occur if the openFT instance has been switched. If the openFT instance is deleted then all running openFT-Script requests are interrupted. They restart when the instance is switched. In the above cases, processing waits for approximately 2 seconds for the end of the activity after interruption of the request. In the case of lengthy *executeScript* activities this may not be enough, with the result that this openFT-Script request is aborted with an error when a restart attempt is made.

2.4 Running an Ftscript

When an Ftscript is run, each activity passes through the following states:

– initialization
– execution
– end
– (error)

Initialization

The context is provided.

Execution

In the case of external activities, the openFT functionality is executed.
In the case of internal activities, the corresponding statement is executed.

If an error occurs during the execution of an activity than an error (or *fault*) is output together with an error code. The activity switches to the "error" state.

End

The end of the activity is reached if execution is terminated without an error. Data may be displayed in the higher-level context.

The activity is terminated. Processing continues with the next activity.
If there are no further activities then the Ftscript is terminated.

Error

The "error" state may be caused by:

– an error occurring during the execution of the activity itself
– an error in a child activity which is not intercepted by a *faulthandler*

A suitable *faulthandler* is searched for in the current context (see section “faulthandler” on page 68). The activity is replaced by the content of the *faulthandler*. In this case, the context objects of the activity are displayed in the *faulthandler*. All the activity's child activities are aborted and their contexts are lost. If these child activities have started file transfer requests then these are also aborted.

If no suitable *faulthandler* is found then the error is passed to the parent activity. The parent activity switches to the "error" state.

If no suitable *faulthandler* is found in any of the higher-level activities then the entire Ftscript is terminated.

Diagnostic information

You can activate a trace in order to conduct a precise analysis of the Ftscript run (including restart, see section “Restart” on page 25):

```
ftscript -t <Ftscript file name>
```

The trace logs every action in the request.

3 openFT-Script Commands

The openFT-Script commands are used to start and administer openFT-Script requests. The requests themselves are stored in a text file in the form of XML statements. These XML statements are described beginning on page 51.

3.1 Overview of the openFT-Script commands

Starting and ending openFT-Script requests

ftscript	Starts an openFT-Script request
ftcans	Cancels an openFT-Script request
ftdels	Deletes an openFT-Script request

Displaying openFT-Script requests and openFT-Script activities

ftshws	Displays openFT-Script requests
ftshwact	Displays the activities of an openFT-Script request

FT administrators can also use the *ftsetjava* command to administer the link to the Java executable, see "openFT System Administrator Manual" and the online help system.

As FT administrator, you can view, cancel and delete all the openFT-Script requests in the system and view the activities associated with all the openFT-Script requests. Users without administrator rights can only administer their own openFT-Script requests.

Variable storage of openFT-Script requests

ftmodsuo	Modify openFT-Script user options
ftshwsuo	Display openFT-Script user options

3.2 ftcans - Cancelling an openFT-Script request

tcans allows you to cancel openFT-Script requests that have not yet been concluded. You can cancel either a specific openFT-Script request or all the openFT-Script requests for a user. This also cancels any file transfer requests started by the specified openFT-Script requests which are currently running. This may take a little time. The status of the openFT-Script request is then set to "cancelled" to prevent any restart.

If the openFT-Script request that is to be cancelled is currently being processed then the following message is output at stderr:

```
ftcans: Cancellation request for ftscript id ftscript id started
```

If the request has been started but not yet processed then the following message is sent to stderr:

```
ftcans: ftscript id ftscript id cancelled.
```

Format for Unix systems

```
ftcans   -h  |
         [ -u=<user ID 1..32> ]
         <ftscriptid> | @a
```

Format for Windows systems

```
ftcans   -h  |
         [ -u=<user ID 1..36> ]
         <ftscriptid> | @a
```

Description

-h Outputs the command syntax on screen. Any specifications after *-h* are ignored.

-u=user ID
User ID under which the search for the openFT-Script request that is to be cancelled is performed.

Only FT administrators may input a user ID.

The default value is the calling party's user ID.

ftscriptid

Identification of the openFT-Script request. This is output if the openFT-Script request is started via an ftscript command.

You can use the wildcard symbols *?* and *** in der *ftscriptid*. This cancels all openFT-Script requests that match the wildcard pattern.

- ? is interpreted as any single character.
- * is interpreted as any number of characters.

If you use wildcards, enclose the *ftscriptid* specification in single quotes so that the wildcard symbols are not interpreted by the shell.

@a means that all the user's openFT-Script requests are to be cancelled.

Return code

0	OK
4	Syntax error
51	Error while outputting an Ftscript user
54	Ftscript ID not found
250	Internal error

3.3 ftdels - Deleting an openFT-Script request

The specified, completed openFT-Script request is deleted from the user's directory or all completed openFT-Script requests are deleted from the user's directory.

No more information is subsequently available for deleted requests. A *ftshws* or *ftshwact* command with the *ftscriptid* of a deleted request is rejected since it no longer exists.

Before an openFT-Script request can be deleted, it must have been completed, i.e. ftshws must indicate the status T, F or C.

i Since *ftcans* is not a synchronous command, it may be necessary to wait for the status C (cancelled) to arise before a subsequent *ftdels*.

If no *ftdels* is issued for an openFT-Script request then this is automatically deleted when the retention period expires.

Format for Unix systems

```
ftdels   -h  |
         [ -u=<user ID 1..32> ]
         <ftscriptid> | @a
```

Format for Windows systems

```
ftdels   -h  |
         [ -u=<user ID 1..36> ]
         <ftscriptid> | @a
```

Description

-h Outputs the command syntax on screen. Any specifications after *-h* are ignored.

-u=user ID

User ID under which the search for the openFT-Script request that is to be deleted is performed.

Only FT administrators may input a user ID.

The default value is the calling party's user ID.

ftscriptid

Identification of the openFT-Script request. This is output when the openFT-Script request is started via an *ftscript* command.

You can use the wildcard symbols *?* and *** in der *ftscriptid*. This deletes all openFT-Script requests that match the wildcard pattern.

?	is interpreted as any single character.
*	is interpreted as any number of characters.

If you use wildcards, enclose the *ftscriptid* specification in single quotes so that the wildcard symbols are not interpreted by the shell.

@a means that all the user's openFT-Script completed requests are to be deleted.

Return code

0	OK
4	Syntax error
51	Error while outputting an Ftscript user
54	Ftscript ID not found
56	openFT-Script has not completed
250	Internal error

3.4 ftmodsuo - Modifying openFT-Script user options

As of openFT V12, users are able to specify where their openFT-Script requests are to be stored. openFT-Script creates the subdirectory *.openFT/<instance>/script* or *.openFT\<instance>\script* in the specified working directory and stores openFT-Script requests in it. The user in question then has write permissions for the subdirectory and it cannot be accessed by other users.

You use the *ftmodsuo* command to specify the directory in which the openFT-Script requests are to be stored. However, you can only do this if no openFT-Script is running and there are no current openFT-Script requests for the user. If necessary, you may have to cancel your running openFT-Script requests with *ftcans* and delete terminated openFT-Script requests with *ftdels*. The command is also rejected if another *ftmodsuo* command for the specification of an openFT-Script working directory is currently running under the same user ID.

Format

```
ftmodsuo -h |
         [ -wd=[ <directory name 1..128> ] ]
```

Description

-h Outputs the command syntax on screen. Any specifications after *-h* are ignored.

-wd Absolute or relative path name of the working directory in which the subdirectory for the user's openFT-Script requests is to be created.

-wd= resets the working directory to the default value, i.e. the user's home directory.

ftmodsuo can also be specified without parameters but does nothing.

Return code

0	OK
4	Syntax error (e.g. the name of the working directory is too long)
15	openFT is not authorized to process requests for this user (e.g. password not set on access to home directory)
69	File access error (*Prelock.lck/UserLock.lck* in *FtscriptWorkdir*)
79	openFT-Script interpreter or other *ftmodsuo* is running. Command aborted
80	Current openFT-Script requests are present. Command aborted
81	Old openFT-Script request not accessible
88	Subdirectories cannot be created in the openFT-Script working directory. Meaning: The directory *<wd>/.openFT/<instance name>/script* or *<wd>\.openFT\<instance name>\script* could not be created, for example due to the absence of write access permission or because a physical error occurred.
90	Working directory does not exist. Command aborted
91	Warning: The previous working directory could not be checked

3.5 ftshwsuo - Displaying openFT-Script user options

You use the *ftshwsuo* command to display the directory in which the openFT-Script requests are to be stored.

Format for Unix systems

```
ftshwsuo -h |
          [ -csv ]
          [ -u=<user ID 1..32 | @a ]
```

Format for Windows systems

```
ftshwsuo -h |
          [ -csv ]
          [ -u=<user ID 1..36 | @a ]
```

Description

-h Outputs the command syntax on screen. Any specifications after *-h* are ignored.

-csv The information is output in CSV format. If you do not specify *-csv* then the information is output in table format.

-u=user ID| **@a**
Only for the FT administrator

User ID whose openFT-Script options are to be displayed:
@a means that the openFT-Script options of all active openFT-Script users as well as of all openFT-Script users who have a working directory other than the default openFT-Script working directory are to be displayed.

Output in table format

User	FtscriptWorkdir
<user>	<path name>

<user>
User ID

<path name>
Designates the name of the openFT-Script working directory that the user has set with *ftmodsuo* without the subdirectory names created by openFT-Script.

If the user has not set any special working directory then the name of his or her home directory is output since this is the openFT-Script directory by default and is used to store the openFT-Script requests.

Output in CSV format

Column	Type	Values
User	String	User ID
FtscriptWorkdir	String	Name of the openFT-Script working directory

Return code

0	OK
4	Syntax error

3.6 ftscript - Starting an openFT-Script request

The *ftscript* command checks the specified script file and executes the statements it includes. The script file must contain a valid XML document which corresponds to the schema for the openFT-Script interface. It must also be possible to read the file using the caller's ID. The maximum number of users who may be owner of openFT-Script requests is 1024. This includes requests that are terminated but not yet deleted.

If errors occur during verification then the script file is not started and the errors are output at stderr.

If the script file starts correctly then the following message is output at stderr:

`ftscript: started successfully. Id:` *ftscript id*

Information about the openFT-Script request is stored in the internal openFT user memory during execution and through to expiry of the retention period. As a consequence, users can view the output *ftscript id* in order to obtain information about the status and success of the operation.

ftscript is restartable, i.e. the processing of the openFT-Script request is ensured even after a system failure.

Format

```
ftscript  -h  |
          [ -t ]
          <Ftscript file name>
```

Description

-h Outputs the command syntax on screen. Any specifications after *-h* are ignored.

-t Diagnostic information (a trace) is created.

Ftscript file name
Name of the script file which contains the XML statements for the openFT-Script request that is to be run.

Return code

0	OK
4	Syntax error
50	Ftscript process could not be started
52	Maximum number of Ftscript users (1024) exceeded
55	Ftscript ID not found
250	Internal error

3.7 ftshwact - Displaying the activity associated with an openFT-Script request

Outputs information about the individual openFT-Script requests.

Format for Unix systems

```
ftshwact   -h  |
          [ -csv]
          [ -a=<ID of the activity> | -d=<Level depth 1...> | -c=<Chapter> ]
          [ -st=[W]|[R]|[T]|[F]|[K]|[D]|[C] ]
          [ -u=<user ID 1..32> ]
            <ftscriptid>
```

Format for Windows systems

```
ftshwact   -h  |
          [ -csv]
          [ -a=<ID of the activity> | -d=<Level depth 1...> | -c=<Chapter> ]
          [ -st=[W]|[R]|[T]|[F]|[K]|[D]|[C] ]
          [ -u=<user ID 1..36> ]
            <ftscriptid>
```

Description

-h Outputs the command syntax on screen. Any specifications after *-h* are ignored.

-csv The information is output in CSV format. If you do not specify *-csv* then the information is output in table format.

-a=ID of the activity

Only the specified activity is displayed.

You may also indicate a specific instruction in a request.

An activity's ID can be determined using a preceding ftshwact command (without the *-a* option). This means that you can view the status of the activity later.

-d=Level depth
Depth of the levels to be displayed.

All activities whose *activity ID* is not greater than the specified level number are displayed. The level number is the number of index numbers separated by dots.

Examples:
from a request with activity IDs 1, 1.2, 1.2(1).1, 1.2(1).2, 1.2(2).1, 1.2(2).2 and1.3 the option *-d*=2 selects the activities with the activity IDs 1, 1.2 and 1.3.

-c=Chapter
Chapter corresponding to the activities to be displayed.

Those activities are output that are a level below the activity with the activity ID specified as the chapter.

In the above example, these are -c=1: 1.2 and 1.3; for -c=1.2: 1.2(1).1, 1.2(1).2, 1.2(2).1 and 1.2(2).2.

-st=[**W**][**R**][**T**][**F**][**K**][**D**][**C**]
Display activities with the specified status. You can specify multiple statuses one after the other, e.g. *-st=WRT.*

Activity 1 is always output since it displays the execution status of the entire script.

-u=user ID
User ID under which the specified request is searched for.

Only FT administrators may input a user ID.

The default value is the calling party's user ID.

ftscriptid
Identification of the openFT-Script request. This is output if the openFT-Script request is started via an *ftscript* command.

You must specify precisely one openFT-Script request. Wildcard syntax is not supported.

Return code

0	OK
4	Syntax error
51	Error while outputting an Ftscript user
53	Ftscript section not found
54	Ftscript ID not found
250	Internal error

Description of the output

Output is possible in tabular form and in CSV format.

It should be noted that for activities which have not yet been started, the output from the *ftshwact* command is usually incomplete since the references present in the request have not yet been resolved and it is not therefore possible to enter all the desired output values. In particular, file and directory names in reference specifications are not fixed until runtime since they may be dependent on the operating system.

Output in table format

The processing level of the activities is displayed in four columns:

Id Unique identification of the activity within the request. This can be converted into an Xpath which mirrors the position of the activity in the tree which is statically predefined by the XML script.

Dynamic information is simply added for the *foreach* nodes (sequence number in the *foreach* loop).

For more detailed information, see the description of the XML statements for the openFT-Script interface.

Sta Status of the statement. The following status identifiers are possible:

W (waiting)	The activity has not yet been started.
R (running)	The activity has been started but has not yet been terminated.
T (terminated)	The activity has been terminated without errors.
F (failure)	The activity has been terminated with an error.
K (killed)	The activity was cancelled by means of a faulthandler or an *ftcans* command.
D (dead)	The activity no longer starts due to a previous error.

In the case of the *ftscript* activity (first activity in an openFT-Script request), a distinction is made between the following statuses:

I (interrupted)	The request was interrupted, e.g. due to a system crash.
C (cancelled)	The request was cancelled with *ftcans*.
X (cancelling)	The request is currently being cancelled due to an *ftcans* command.
F (failure)	Is only displayed for the *ftscript* activity if the error was not handled by a *faulthandler*.

In the case of activities with the status F and *faulthandler* activities, the cause of the error is output in clear text in an additional line.

Activity
Activity name. The names are based on the openFT-Script language but may be truncated in some cases, e.g. *faulthdlr* instead of *faulthandler*.

foreach is designated in accordance with the value of the execute attribute as *foreach_P* (parallel) or *foreach_S* (sequential).

TransferFile is designated as *sendFile* or *rcvFile* (=receive File) depending on the direction of transfer.

ActivityObject
The content of this column depends on the activity in question, see the table below.

Activity	**ActivityObject**	**Meaning**
ftscript	<scriptPath>	Complete path name of the original file with the XML statements.
empty	-	
foreach_P	<contextObject>	context object which assumes the value of the current list element
foreach_S	as foreach_P	
parallel	-	
sequence	-	
sendFile	Specifies the remote file in the following form:	
	<partner>!<file name>	Partner with file name if both are known.
	*unknown!<file name>	if the partner is not yet known.
	*unknown!*unknown	if both are not yet known.
	<partner>!*ref(<contextId>)	if *contextId = foreach contextObject* and the resolution is not yet known because it has not yet been passed through.

Activity	**ActivityObject**	**Meaning**
sendFile (*cont.*)	<file name>	in the case of requests which have already been started, this is the name specified in the FT request. In the case of requests which have not yet been started, this name is derived from the operating system-specific name specified in the XML file (e.g. unixName) and extended by the directory specifications.
rcvFile	as sendFile.	
deleteFile	specifies the remote file as in sendFile (with partner), if the file is local, without partner:	
	<file name>	like *sendFile*, is determined from the FT request in the case of requests that have already been started, and from the XML file in the case of requests that have not yet started. A local file name would be output as an absolute file name in the case of a started request and as a relative path name in the case of an as yet unstarted request.
	*unknown!<file name>	if it is not known if the file is local when a file object is referenced.
createDir	<partner>! <directory-name>	Partner with directory name if both are known.
	*unknown! <directory-name>	if the partner is not yet known.
	*unknown!*unknown	if both are not yet known.
	<partner>! *ref(<contextID>)	if *contexId = foreach contextObject* and the resolution is not yet known because it has not yet been passed through.

Activity	**ActivityObject**	**Meaning**
createDir (*cont.*)	<directory-name>	if the directory is local. In this case, as for *sendFile*, the name for already started requests is determined from the FT request and for requests which have not yet been started, from the specifications in the XML file. A local file name would be output as an absolute file name in the case of a started request and as a relative path name in the case of an as yet unstarted request.
deleteDir	as createDir.	
listDir	as createDir.	
execScript	32 characters.	Contains the first 32 characters of the command that is to be executed. For security reasons, the user should make sure that the first 32 characters do not contain any confidential parameters.
fault	<faultcode>	Error code specified by the user.
faulthdl	<triggering activity id>: <special faultcode>; <general faultcode>	

Output in CSV format

```
Id;State;Activity;ActivityObject;Partner;AddInfo;nrElements;
StartTime;Error
```

The output contains the following information:

Id	See table format on page 42.
State	See table format on page 42.
Activity	See table format on page 43.
ActivityObject	See table format, enclosed in double quotes, otherwise: - the path name is output without partner specifications - only the *faultcodes* are output for the *faulthandler* activity.
Partner	In the case of path-related activities, the partner or partner specification that would be present in front of the path name in table format, enclosed in double quotes. Otherwise empty.
AddInfo	For *sendFile* and *rcvFile*: TID, enclosed in double quotes if the activity has already started. Otherwise empty. For *faulthdl*, the triggering *activity-Id* enclosed in double quotes. Otherwise empty.
nrElements	In the case of a started *foreach*: number of loop passes. In the case of a started *parallel* or *sequence*: number of sub-activities.
StartTime	Start time in the format yyyy-mm-dd hh:mm:ss
Error	In the case of requests with the status F, case of error in clear text enclosed by double quotes. Otherwise empty.

3.8 ftshws - Displaying openFT-Script requests

Outputs information about the status of a user's openFT-Script requests. You can also specify a *ftscriptid* in order to select a specific openFT-Script request.

Format for Unix systems

```
ftshws  -h  |
        [ -csv]
        [ -t]
        [ -v]
        [ -st=[W]|[R]|[T]|[F]|[I]|[C]|[X] ]
        [ -u=<user ID 1..32> | @a ]
        [<ftscriptid>]
```

Format for Windows systems

```
ftshws  -h  |
        [ -csv]
        [ -t]
        [ -v]
        [ -st=[W]|[R]|[T]|[F]|[I]|[C]|[X] ]
        [ -u=<user ID 1..36> | @a ]
        [<ftscriptid>]
```

Description

-h Outputs the command syntax on screen. Any specifications after *-h* are ignored.

-csv The information is output in CSV format. If you do not specify *-csv* then the information is output in table format.

-t The openFT-Script requests are displayed sorted on generation time, beginning with the last request.

By default, the requests are displayed in alphabetical order.

-v Diagnostic information is also output (verbose).

If *-v* is specified then, in the case of openFT-Script requests which terminate with an error, the cause of the error is output in a second line after the tabular information.

In CSV format, the *-v* option is ignored.

--st=[**W**][**R**][**T**][**F**][**I**][**C**][**X**]

displays openFT-Script requests with the specified status, see *Sta* field in "Output in table format" on page 49.

You can specify multiple statuses one after the other, e.g. *-st=WRT*.

-u=user ID | **@a**

User ID for which openFT-Script requests are output or under which the specified request is searched for.

Only administrators may specify or *@a* (all user IDs).

The default value is the calling party's user ID.

ftscriptid

Identification of the openFT-Script request. This is output if the openFT-Script request is started via an *ftscript* command.

You can use the wildcard symbols *?* and *** in der *ftscriptid*. This outputs all openFT-Script requests that match the wildcard pattern.

?	is interpreted as any single character.
*	is interpreted as any number of characters.

If you use wildcards, enclose the *ftscriptid* specification in single quotes so that the wildcard symbols are not interpreted by the shell.

By default, if you do not specify *ftscriptid,* all the user's openFT-Script requests are displayed.

Return code

0	OK
4	Syntax error
51	Error while outputting an Ftscript user
54	Ftscript ID not found
250	Internal error

Output in table format

The processing level of the openFT-Script requests is displayed in four columns:

User	User ID under which the request was started.
Ftscriptid	Unique identification of the request. The identification is returned by the *ftscript* command.
Sta	Indicates the processing status, where:

W (waiting)	The request has not yet been started.
R (running)	The request has been started but has not yet been terminated.
T (terminated)	The request has been terminated without errors.
F (failure)	The request has been terminated with errors.
I (interrupted)	The request was interrupted, e.g. due to a system crash.
C (cancelled)	The request was cancelled with an *ftcans* command.
X (cancelling)	The request is currently being cancelled due to an *ftcans* command.

FtscriptFileName	Path name of the script file.

If the status F and the *-v* option are specified then the cause of the error is output in clear text in another column.

Output in CSV format

```
User;Ftscriptid;State;CreationTime;FtscriptFileName;Error;
```

The output contains the following information:

User	User ID under which the request was started.
Ftscriptid	Unique identification of the request. The identification is returned by the *ftscript* command.
State	See table format (Sta).
CreationTime	Time at which the openFT-Script request was created, in the format yyyy-mm-dd hh:mm:ss.
FtscriptFileName	Path name of the script file.
Error	Cause of error in clear text in the case of openFT-Script requests with status F, otherwise empty.

User, *Ftscriptid*, *FtscriptFileName* and, if applicable, *Error* are output enclosed in double quotes.

4 openFT-Script statements

This section describes the individual openFT-Script statements in alphabetical order:

– The use of the statement.

– The existing restrictions.

– The format describes the syntax of the statement. For an explanation of the syntax, see the following section “Syntax of the openFT-Script statements”.

– The statement's available attributes, their values and the meaning of these values. Optional attributes are indicated by a "?".

– The examples illustrate the use of the statement.

4.1 Syntax of the openFT-Script statements

The openFT-Script statements are described in the following sections. The syntax of the openFT-Script statements is specified in the "Format" section in the description of each statement. The following symbols are used:

Symbol	**Meaning**
`a?`	No element or an element a. Optional attributes are also indicated by a "?".
`a*`	No element or any number of elements a.
`a+`	One element a or multiple elements a.
`|` `a|b+`	Either ... or ... Either (exactly) one element a or one or more elements b.
`<...>`	The current element with its attributes is presented in angle brackets. Elements can be combined to form an activity. Such combined elements are printed without the angle brackets.

4.2 baseDir

You use *baseDir* to define a base directory for the following openFT-Script statements:

– *createDirectory*
  The directory which you create with *createDirectory* is set up under the base directory (see section “createDirectory” on page 55).

– *deleteDirectory*
  The directory which you delete with *deleteDirectory* is deleted under the base directory (see section “deleteDirectory” on page 57).

– *deleteFile*
  The file which you delete with *deleteFile* is deleted under the base directory (see section “deleteFile” on page 59).

– *listDirectory*
  The files or directories under the base directory are listed. The base directory itself is not listed (see section “listDirectory” on page 79).

You can only use *baseDir* with the openFT-Script statements which are listed above.

Format

See the corresponding openFT-Script statement.

Attributes

Name	**Value**	**Meaning**
`dirnames`	See section “Directory name attributes” on page 21.	

Examples

See:
– section “createDirectory” on page 55.
– section “deleteDirectory” on page 57.
– section “listDirectory” on page 79.

4.3 comment

You use *comment* to enter a comment text for the element in question.

You can use comments to describe the scripts.

You use XML comments (<!--...-->) to make internal comments regarding the scripts.

Format

```
<comment>
  text
</comment>
```

4.4 context

You use *context* to define a context and *faulthandlers* for an activity.

Context objects are objects which can be used in the same or in lower-level activities when referenced using the attribute *ref* or *listRef*. They may be of type *autoDataSpec*, *directory*, *file*, *list*, *partner* or *script*.

A context exists for every activity. If no context element is specified the an empty context is explicitly created.

All the context objects and *faulthandlers* of the higher-level contexts are visible provided that they are not hidden by context objects or *faulthandlers* with the same name.

Format

```
<context>
 ContextObject*
 faulthandler?
</context>
```

4.5 createDirectory

You use *createDirectory* to create a directory If you do not specify a partner then the directory is created under the local user ID.

You can use *baseDir* (see section “baseDir” on page 52) to define a base directory under which the specified directory is created.

The length of the directory name (length of *baseDir* plus length of *dirnames*) is limited and depends on the openFT version. The length is the number of characters plus 1 character if *baseDir* does not end with a "/".

i You should note the response on a restart (see section “Restart” on page 25).

Restrictions

If you specify a directory path in the name attribute then all the directories down to the lowest level must already exist.

Format

```
<createDirectory ref?="ID" faultIfExists?="yes|no" dirnames >
  comment?
  context?
  partner?
  baseDir?
</createDirectory>
```

Attributes

Name	Value	Meaning
`ref?`	string	Reference to a directory context object
`faultIfExists?`	yes | no	The default value is *no*. If the directory exists then *createDirectory* is terminated without error. If *yes* is specified then *createDirectory* is aborted with the error code *ft_exists* if the directory already exists. On a restart (see section “Restart” on page 25), *createDirectory* is aborted with the error code *ft_recoveryCreateDirectory* if the directory exists This may also occur if the instance is switched.
`dirnames`	See section “Directory name attributes” on page 21.	

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <directory id="myDir" name="myTmp"/>
  </context>
  <createDirectory ref="myDir">
    <partner name="UnixP_1">
      <transferAdmission>
        <ftacAdmission ftacAdmission="FTACADM1"/>
      </transferAdmission>
    </partner>
    <baseDir name="frg_eis_03"/>
  </createDirectory>
</ftscript>
```

Creates the directory *myTmp* for the FTAC transfer admission *FTACADM1* in the directory *frg_eis_03* on the computer *UnixP_1*.

createDirectory is terminated without error if the directory already exists.
Once *createDirectory* has run, the directory exists.

frg_eis_03 is specified as *baseDir*. The directory that is to be created is referenced. The entire directory path (*baseDir* + *name*) is *frg_eis_03/myTmp* and consists of 16 characters.

If the desired directory *frg_eis_03/myTmp* cannot be created, for example because the path *frg_eis_03* does not exist then *createDirectory* is terminated with the error code *ft_cantCreate*.

A further example of *createDirectory* can be found in section “foreach” on page 73.

4.6 deleteDirectory

You use *deleteDirectory* to delete a directory. If you do not specify a partner then the directory is deleted under the local user ID.

You use *baseDir* to specify a base directory (see section “directory” on page 61) under which the directory that is to be deleted is to be searched for. The base directory name and the name specified with the *dirnames* attribute are combined to form the directory name.

The length of the directory name (length of *baseDir* plus length of *dirnames*) is limited and depends on the openFT version. The length is the number of characters plus 1 character if *baseDir* does not end with a "/".

i You should note the response on a restart (see section “Restart” on page 25).

Restrictions

The directory that is to be deleted must be empty.

Format

```
<deleteDirectory ref?="ID" faultIfNotExists?="yes|no" dirnames >
  comment?
  context?
  partner?
  baseDir?
</deleteDirectory>
```

Attributes

Name	Value	Meaning
`ref?`	string	Reference to a directory context object
`faultIfNotExists?`	`yes` | `no`	The default value is *no*. If the directory does not exist then *deleteDirectory* is terminated without error. If *yes* is specified then *deleteDirectory* is terminated with the error code *ft_notExists* if the directory does not exist. On a restart (see section “Restart” on page 25), *deleteDirectory* is terminated with the error code *ft_recoveryFailed*. This may also occur if the instance is switched.
`dirnames`	See section “Directory name attributes” on page 21.	

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <directory id="myDir" name="myTmp"/>
  </context>
  <deleteDirectory ref="myDir">
    <partner name="UnixP_1" systemType="unix">
      <transferAdmission>
        <ftacAdmission ftacAdmission="FTACADM1"/>
      </transferAdmission>
    </partner>
    <baseDir name="frg_eis_04"/>
  </deleteDirectory>
</ftscript>
```

Deletes the directory *myTmp* for the FTAC transfer admission *FTACADM1* on the computer *UnixP_1*.

deleteDirectory is terminated without error if the directory does not exist or has already been deleted.

frg_eis_04 is specified as *baseDir*. The directory that is to be deleted is referenced. The entire directory path (*baseDir* + *name*) is *frg_eis_04/myTmp* and consists of 16 characters.

4.7 deleteFile

You use *deleteFile* to delete a file. If you do not specify a partner then the file is deleted under the local user ID.

The name of the file that is to be deleted consists of the directory name specified with *directory* (see section “directory” on page 61) and the name specified with the *filenames* attribute.

If you want to delete all the files in a directory, you should use *listDirectory* (see section “listDirectory” on page 79) together with *foreach* (see section “foreach” on page 73).

The length of the file name (length of *directory* plus length of *filenames*) is limited and depends on the openFT version. The length is the number of characters plus 1 character if *directory* does not end with a "/"

i You should note the response on a restart (see section “Restart” on page 25).

Format

```
<deleteFile ref?="ID" faultIfNotExists?="yes|no" filenames >
  comment?
  context?
  partner?
  directory?
</deleteFile>
```

Attributes

Name	Value	Meaning
`ref?`	string	Reference to a file context object
`faultIfNotExists?`	yes | no	The default value is *no*. If the file does not exist then this is not considered to be an error and Ftscript processing is continued. If *yes* is specified then the Ftscript request is aborted with *ft_notExists* if the file does not exist. On a restart (see section “Restart” on page 25), the Ftscript request is aborted with *ft_recoveryFailed*. This may also occur if the instance is switched.
`filenames`	See section “File name attributes” on page 20.	

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <deleteFile name="hugo.trash">
    <partner name="UnixP_1" systemType="unix">
      <transferAdmission>
        <ftacAdmission ftacAdmission="FTACADM1"/>
      </transferAdmission>
    </partner>
    <directory name="frg_eis_05"/>
  </deleteFile>
</ftscript>
```

Deletes the file *hugo.trash* for the FTAC transfer admission *FTACADM1* in the directory *frg_eis_05* on the computer *UnixP_1*.

Errors are not handled in this example and result in the Ftscript being deleted.

4.8 directory

You use *directory* to define a directory path.

directory is always a child element of an Ftscript activity. If a remote directory is intended, specify the partner as a further subelement of the activity.

Format

```
<directory id="ID" ref?="ref" dirnames >
  comment?
</directory>
```

Attributes

Name	Value	Meaning
`id`	string	A unique ID in the current context. The context object is referenced under this ID.
`ref?`	string	Name of the directory context object. Data which is not present here is taken over from the directory context object after de-referencing.
`dirnames`	See section “Directory name attributes” on page 21.	

4.9 empty

The activity *empty* does nothing but is required for formal purposes in order to intercept an error or other actions in a *faulthandler* (see section “faulthandler” on page 68).

Format

```
<empty>
  comment?
</empty>
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <transferFile>
    <context>
      <faulthandler>
        <default>
          <empty/>
        </default>
      </faulthandler>
    </context>
    <fromRemoteFile name="pack1.bin">
      <partner name="someHost">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM1"/>
        </transferAdmission>
      </partner>
      <directory name="frg_eis_06"/>
    </fromRemoteFile>
    <toLocalFile name="target.bin">
      <directory name="frg_eis_06"/>
    </toLocalFile>
  </transferFile>
</ftscript>
```

If errors occur in *transferFile* then execution of the openFT-Script request continues nevertheless, i.e. the Ftscript request is terminated with status T.

In the case of "severe" errors (see section “"Severe" Ftscript error codes” on page 24), the openFT-Script request is terminated with the corresponding error code since the *default faulthandler* is ineffective.

4.10 executeScript

You use *executeScript* to run a script.

The script is executed in the target system. If you do not specify a partner then the script is executed on the local system under the user ID of the user who called the Ftscript.

The following command interpreters are used:

Operating system	**Command interpreter**
Windows	System call, i.e. an executable file with the specified name is searched for. E.g. to execute a shell command, enter *cmd /c*.
Unix system	/bin/sh -c
z/OS	TSO
BS2000	SDF

Restrictions

1. A script (e.g. bs2000Script, unixScript) may only occupy one line and is limited to 500 characters in length.
2. You must specify a script which is not empty for the addressed operating system.

Format

```
<executeScript ref?="ID" repeatable?="true|false" >
  comment?
  context?
  script?
  bs2000Script?
  unixScript?
  windowsScript?
  zosScript?
  partner?
</executeScript>
```

Attributes

Name	Value	Meaning
`ref?`	string	Reference to a script context object
`repeatable?`	`true` | `false`	The default value is *true*. The script may be repeated on a restart. If *false* is specified: The script may not be repeated on a restart. On a restart, the *executeScript* activity is aborted with the error code *ft_resumeForbidden* if it is not possible to determine whether this script has been fully processed. It is only possible to switch the openFT instance during script execution in the case of scripts with short runtimes (see section “Restart” on page 25).

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <list id="partnerList">
      <partner name="WindowsP_1" systemType="windows">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM1"/>
        </transferAdmission>
      </partner>
      <partner name="UnixP_1" systemType="unix">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM2"/>
        </transferAdmission>
      </partner>
    </list>
  </context>
  <foreach listRef="partnerList" selectType="partner"
   contextObject="partner">
    <executeScript>
      <unixScript><![CDATA[echo hello unix >frg_eis_07/demo.txt]]>
      </unixScript>
      <windowsScript><![CDATA[cmd /c echo 'hello windows'
           >frg_eis_07\demo.txt]]>
      </windowsScript>
      <partner ref="partner"/>
    </executeScript>
  </foreach>
</ftscript>
```

An *executeScript* is run on the computers in the list *partnerList*.
Corresponding operating-specific scripts are executed depending on the operating system in question. It is important to specify the operating system in the partner definition.

4.11 fault

You use *fault* to cancel the parent activity (and all its running child activities) with a user-defined error code and continue execution in the corresponding *faulthandler*. The *faulthandler* of the parent activity is processed first.

The error is intercepted with the appropriate *faulthandler* (*default* or *case*) (see section “faulthandler” on page 68). The activity associated with the *faulthandler* is executed.

All file transfer requests that were started by the parent activity and are still running are also cancelled (*ft_cancel*). This may result in the execution of the *remoteFailureScript* (see section “remoteFailureScript” on page 108).

If it is not possible to assign a *faulthandler* to the error code then the entire script is aborted.

Restrictions

The error code must not start with "ft_". These error codes are reserved for openFT-Script.

Format

```
<fault code="faultcode">
  comment?
</fault>
```

Attributes

Name	Value	Meaning
`code`	Text	The error code which can be intercepted in a *faulthandler*.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <parallel>
    <context>
      <partner id="remote" name="UnixP_1">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM1"/>
        </transferAdmission>
      </partner>
      <faulthandler>
        <case code="intercept">
          <deleteFile name="target1.bin">
            <partner ref="remote"/>
            <directory name="frg_eis_08NotAvailable"/>
          </deleteFile>
          <deleteFile name="target2.bin">
            <partner ref="remote"/>
            <directory name="frg_eis_08"/>
          </deleteFile>
        </case>
      </faulthandler>
    </context>
    <transferFile>
      <context>
        <faulthandler>
         <default>
            <fault code="intercept"/>
          </default>
        </faulthandler>
      </context>
      <fromLocalFile name=
"W:/openFT/ftscript/Test/data/small/bin.mp3"/>
      <toRemoteFile name="target1.bin">
        <partner ref="remote"/>
        <directory name="frg_eis_08NotAvailable"/>
      </toRemoteFile>
    </transferFile>
```

Example (cont.)

```
    <transferFile>
      <context>
        <faulthandler>
          <default>
            <fault code="intercept"/>
          </default>
        </faulthandler>
      </context>
      <fromLocalFile name=
"W:/openFT/ftscript/Test/data/large/bin.mp3"/>
      <toRemoteFile name="target2.bin">
        <partner ref="remote"/>
        <directory name="frg_eis_08"/>
      </toRemoteFile>
    </transferFile>
  </parallel>
</ftscript>
```

Two file transfers are performed in parallel.
If an error occurs during one of the transfers then this is indicated by the error code *intercept*.
This is intercepted in the *faulthandler*.
The other file transfer is cancelled (*ft_cancel*).
The two target files are deleted in the *faulthandler*. Any errors which occur are ignored.

Using this script either both files or neither of the files reach their destination.

i If a *remoteFailureScript* is defined for the file transfer which is cancelled by means of *ft_cancel* then this may continue to run even when the *faulthandler* is already active.

4.12 faulthandler

The *faulthandler* is analyzed if an error occurred in the activity in which it is present or if the activity is switched to the "error" state due to a child activity (see section “Running an Ftscript” on page 26)

The *faulthandler* is used if a *case* activity with the corresponding error code is defined in it or, in the absence of any "severe" error, a *default* activity is defined (see section “"Severe" Ftscript error codes” on page 24).

This *case* or *default* activity then replaces the activity in which the *faulthandler* is located, takes over its context objects and is then executed.

If an error occurs during the execution of the *case* or *default* activity, then the original *faulthandler* is ignored and a suitable *faulthandler* in the activity's own context or that of its parent activities is used.

Format

```
<faulthandler>
  comment?
  (case* default) | (case+ default?)
</faulthandler>
```

Attributes

See section “case” on page 70.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <partner id="RemotePartner" name="D018S011">
      <transferAdmission>
        <ftacAdmission ftacAdmission="p1111111"/>
      </transferAdmission>
    </partner>
  </context>
  <transferFile>
    <context>
      <file id="source" name="source.bin"/>
      <file id="target" name="target.bin">
        <partner ref="RemotePartner"/>
      </file>
      <faulthandler>
        <default>
          <transferFile>
            <context>
              <partner id="RemotePartner" name="D018S022">
                <transferAdmission>
                  <ftacAdmission ftacAdmission="p2222222"/>
                </transferAdmission>
              </partner>
            </context>
            <fromLocalFile ref="source"/>
            <toRemoteFile ref="target"/>
          </transferFile>
        </default>
      </faulthandler>
    </context>
    <fromLocalFile ref="source"/>
    <toRemoteFile ref="target"/>
  </transferFile>
</ftscript>
```

In this example, the local file *source.bin* is to be copied to *D018S011/taget.bin*.
If an implicit error occurs in *transferFile* then the file is copied to *D018S022/target.bin*. This overlays the context object with the Id *Remote Partner*. If another error occurs during this activity then the Ftscript is cancelled.

case

The activity described in *case* is executed if the current error code is found in its list of error codes. The *case* activity replaces the activity in which the *faulthandler* is located.

The context objects of the replaced activity are copied to the context of the *case* activity if still present. The contexts of the child activities (including those in which the error occurred) are no longer accessible.

For information on execution, see section “sequence” on page 90.

Format

```
<case code="codelist" >
  comment?
  context?
  Activity+
</case>
```

Attributes

Name	Value	Meaning
code	codelist	A list of error codes for which this *case* activity is to be executed. The individual error codes are separated from one another by spaces. Here you can use “normal” Ftscript error codes (see page 23) and “severe” Ftscript error codes (see page 24) or a user-defined error code that you create with the *fault* activity

default

If the current error code is not found in any of the *case* elements then the *default* activity is executed.

This applies to all error codes with the exception of the "severe" Ftscript error codes (see section “"Severe" Ftscript error codes” on page 24), for which the *default* branch of the *faulthandler* is ignored.
The *default* activity replaces the activity in which the *faulthandler* is located.

The context objects of the replaced activity are copied to the context of the *default* activity if still present. The contexts of the child activities (including those in which the error occurred) are no longer accessible.

For information on execution, see section “sequence” on page 90.

Format

```
<default>
  comment?
  context?
  Activity*
</default>
```

4.13 file

You use *file* to define a file.

Properties of a file:

- The file has a system-specific name.
- The file is located on a concrete system (*partner*) in a concrete directory (*directory*). If no partner is specified then the file is located on the local system.

Restrictions

The same restrictions apply as for *filenames* and *directory* (see sections "File name attributes" on page 20 and "Directory name attributes" on page 21).

Format

```
<file id="ID" ref?="ref" filenames >
  comment?
  partner?
  directory?
</file>
```

Attributes

Name	Value	Meaning
`id`	string	A unique ID in the current context. The context object is referenced under this ID.
`ref?`	string	Name of another file context object.
`filenames`	See section "File name attributes" on page 20.	

4.14 foreach

You use *foreach* to execute a sequence for each element in a list (see section “list” on page 78). *foreach* executes the child elements of each element in the selected list as a sequence.

You can specify whether the sequences are executed one after the other (in the same order as the list elements) or in parallel.

Format

```
<foreach listRef="ID" contextObject="ID" execute?=
"parallel|sequential"
         selectType="file|partner|directory" direction?=
"forward|reverse" >
  comment?
  context?
  Activity+
</foreach
```

Attributes

Name	Value	Meaning
listRef	string	Name of a valid context object of type *list*.
contextObject	string	Name of the *foreach* context object which takes on the value of the current list element. This must not be defined in the *foreach* context. It is defined implicitly. The type of context object corresponds to the type set in the *selectType* attribute.
execute?	parallel | sequential	The default value is *sequential*. The sequences are executed one after the other. If *parallel* is specified then the sequences are started in parallel.
selectType	partner | file | directory	Filters the elements of the specified type from the list. Only the filtered elements are iterated.
direction?	forward | reverse	The default value is *forward*. The list is worked through forwards. If *reverse* is specified then the list is worked through backwards.

Examples

1. Distributing files

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <list id="FileList">
      <file name="bin.mp3"/>
      <file name="text.txt"/>
    </list>
    <list id="HostList">
      <partner name="UnixP_1" systemType="unix">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM1"/>
        </transferAdmission>
      </partner>
      <partner name="WindowsP_1" systemType="windows">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM2"/>
        </transferAdmission>
      </partner>
    </list>
  </context>
  <foreach listRef="HostList" selectType="partner"
           contextObject="partner" execute="parallel">
    <foreach listRef="FileList" selectType="file"
             contextObject="file" execute="parallel">
      <transferFile>
        <fromLocalFile ref="file">
          <directory name="W:/openFT/ftscript/Test/data/large"/>
          <autoDataSpec binPattern="*.mp3" charPattern="*.txt"/>
        </fromLocalFile>
        <toRemoteFile ref="file">
          <partner ref="partner"/>
          <directory name="frg_eis_09"/>
        </toRemoteFile>
      </transferFile>
     </foreach>
  </foreach>
</ftscript>
```

The files *bin.mp3* and *text.txt* are copied to two computers.

In the example, the lists of files and computers are defined as context objects. The file list can also be defined, for example, by means of a *listDirectory* (see section “listDirectory” on page 79).

A double *foreach* sequence is used. The external sequence works through all the computers and the inner sequence works through all the files. The connection to the computers takes place in parallel and the files are also worked through in parallel at each computer.

autoDataSpec differentiates between text and binary files (see section “autoDataSpec” on page 98).

When the script is run, the files are distributed to all the computers.
Since no *faulthandler* was used in the example, the script is terminated with an error.

2. Copying the file tree

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <partner id="remote" name="UnixP_1">
      <transferAdmission>
        <ftacAdmission ftacAdmission="FTACADM1"/>
      </transferAdmission>
    </partner>
  </context>
  <listDirectory name="*//*" listObject="Flist">
    <partner ref="remote"/>
    <baseDir name="frg_eis_11"/>
  </listDirectory>
  <foreach listRef="Flist" selectType="directory"
           contextObject="creDir" execute="sequential">
    <createDirectory ref="creDir">
      <baseDir name="frg_eis_11"/>
    </createDirectory>
  </foreach>
  <foreach listRef="Flist" selectType="file"
           contextObject="file" execute="parallel">
    <transferFile>
      <fromRemoteFile ref="file">
        <partner ref="remote"/>
        <directory name="frg_eis_11"/>
      </fromRemoteFile>
      <toLocalFile ref="file">
        <directory name="frg_eis_11"/>
      </toLocalFile>
    </transferFile>
  </foreach>
</ftscript>
```

In the directory *frg_eis_11* on the computer *UnixP_1*, the file tree **//** is copied to the directory *frg_eis_11* under the local ID.

In the first *foreach* sequence, all the necessary directories are copied sequentially using *createDirectory*.
listDirectory returns the directories *a*, *a/b* and *a/b/c* in sequence for the directory *frg_eis_11/a/b/c* (*frg_eis_11* is itself defined as the base directory with *baseDir*). A directory cannot be created unless the parent directory exists.

In the second *foreach* sequence, the files are copied in parallel since all the target directories are now present.

3. Deleting a file tree

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <partner id="remote" name="UnixP_1">
      <transferAdmission>
        <ftacAdmission ftacAdmission="FTACADM1"/>
      </transferAdmission>
    </partner>
  </context>
  <listDirectory name="frg_eis_10/*//*" listObject="Flist">
    <partner ref="remote"/>
  </listDirectory>
  <foreach listRef="Flist" selectType="file"
           contextObject="delFile" execute="parallel">
    <deleteFile ref="delFile">
      <partner ref="remote"/>
    </deleteFile>
  </foreach>
  <foreach listRef="Flist" selectType="directory"
           contextObject="delDir" execute="sequential"
                          direction="reverse">
    <deleteDirectory ref="delDir">
      <partner ref="remote"/>
    </deleteDirectory>
  </foreach>
</ftscript>
```

In this example, everything in the directory *frg_eis_10* on the computer *UnixP_1* is deleted under the FTAC transfer admission *FTACADM*.

listDirectory (see section "listDirectory" on page 79) is used to determine all the files and directories recursively using the search pattern *//*. The sequence in which the directories are listed corresponds to the sequence required for their generation (i.e. the opposite sequence is required in order to delete them).

In the first *foreach* sequence, all the files are deleted in parallel. Non-existent files are ignored. An error during file deletion results in cancellation of the script.

In the second *foreach* sequence, the empty directories are deleted backwards because the directories to be deleted with *deleteDirectory* must be empty (see section "deleteDirectory" on page 57). Non-existent directories are ignored. Other errors result in the cancellation of the script.

When the script has run, the directory *frg_eis_10* on the computer *UnixP_1* is empty.

4.15 ftscript

ftscript is the root element of the script.

The element always corresponds to a *sequence* activity (see section “sequence” on page 90).

Format

```
<ftscript version="1">
  comment?
  context?
  Activity+
</ftscript>
```

Attributes

Name	Value	Meaning
`version`	`1`	Fixed value describing the Ftscript version.

Example

See any example in the current manual.

4.16 list

A list contains multiple elements of type *partner*, *directory* or *file*.

You can also generate a list using *listDirectory* (see section “listDirectory” on page 79).

foreach permits the iterative processing of the elements in the list (see section “foreach” on page 73).

Format

```
<list id="ID" >
 comment?
 ( partner | directory | file )*
</list>
```

Attributes

Name	Value	Meaning
`id`	string	A unique ID in the current context. The context object is referenced under this ID.

Example

See section “foreach” on page 73.

4.17 listDirectory

You use *listDirectory* to list the files and directories. The located file or directory names are combined in a list. The list is displayed in the parent context in a context object with the specified *listObject-Id*. The context object is available there after execution of the *listDirectory* activity.
If an error occurs during the execution of *listDirectory* then the object is not available.

i A *listDirectory* as a direct child element of a *parallel* activity does not return a usable event list (see section “parallel” on page 82).

You can use *baseDir* (see section “baseDir” on page 52) to define a base directory from which *listDirectory* is run.
The base directory itself is not listed in the result.

File or directory names may be a maximum of 512 characters in length. This is checked before the Ftscript is run. The length of the resulting file or directory name (*baseDir* and *dirnames* or *baseDir* + *filenames*) is limited. The length is the number of characters plus 1 character if *baseDir* does not end with a "/". The check is performed while the Ftscript is being run.
For further information, see section “Specifying file and directory names” on page 19

i You can use the wildcard symbol **//** to list a directory tree in full.

Format

```
<listDirectory listObject="ID" dirnames >
  comment?
  context?
  partner?
  baseDir?
</listDirectory>
```

Attributes

Name	Value	Meaning
`ref?`	string	Reference to a directory context object
`listObject`	string	Name of the *list* context object which is displayed in the parent context. No context element with this name may exist.
`dirnames`	See section “Directory name attributes” on page 21. To list a directory tree in full, you can specify the wildcard symbol *//* at the end of the name.	

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <list id="RemoteHostList">
      <partner name="UnixP_1">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM1"/>
        </transferAdmission>
      </partner>
      <partner name="WindowsP_1">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM2"/>
        </transferAdmission>
      </partner>
    </list>
  </context>
  <listDirectory listObject="FileList">
    <baseDir name="W:/openFT/ftscript/Test/multi"/>
  </listDirectory>
  <foreach listRef="RemoteHostList" selectType="partner"
           contextObject="RemoteHost" execute="parallel">
    <foreach listRef="FileList" selectType="file"
             contextObject="File" execute="parallel">
      <transferFile>
        <fromLocalFile ref="File">
          <directory name="W:/openFT/ftscript/Test/multi"/>
        </fromLocalFile>
        <toRemoteFile ref="File">
          <partner ref="RemoteHost"/>
          <directory name="frg_eis_12/local/bin"/>
        </toRemoteFile>
      </transferFile>
    </foreach>
  </foreach>
</ftscript>
```

In the example, all the files from the local directory *W:/openFT/ftscript/Test/multi/* are written to the context object *FileList*.
The list only contains the file names, not the *baseDir* directory.

The files are distributed in parallel to *frg_eis_12/local/bin/* on all the computers in the *RemoteHostList*.

The context object *FileList* from the *listDirectory* activity is displayed in the context of the parent element (in the example, *ftscript*).

4.18 parallel

You can specify *parallel* to run all the activities "simultaneously" and independently of one another. However, truly simultaneous execution (for example, as in the case of time slicing) is not implemented.

The *parallel* activity is terminated when all the child activities have terminated. If one child activity outputs an error then child activities that are still running are cancelled.

The results of any given child activity are not visible in the other child activities.

The transfer of context objects to the parent context is not planned at present. Consequently, context objects which arise during the *parallel* activity are discarded.

i A *listDirectory* as a direct child element of a *parallel* activity therefore does not return a usable event list (see section “listDirectory” on page 79).

parallel refers to the parallel processing of activities by openFT-Script. The maximum number of file transfers that actually run in parallel is determined by the openFT connection limit and process limit (see the openFT manual “Installation and Administration”). To save resources, openFT-Script restricts the number of requests in the request queue in order to ensure that this connection limit is not exceeded. In the case of *parallel*, the requests can be processed in any order.

Synchronous activities such as *deleteScript* are also not necessarily all started at the same time within the framework of a *parallel* activity. Instead, the maximum number is 200, so that not too many threads have to be established.

i The openFT connection limit and process limit operating parameters control whether requests in the request queue are processed simultaneously or sequentially and consequently influence the actual level of parallel execution as well as the performance and resource consumption of your openFT-Script request.

Format

```
<parallel>
  comment?
  context?
  Activity+
</parallel>
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <parallel>
    <transferFile>
      <fromLocalFile name=
       "W:/openFT/ftscript/Test/data/large/text.txt"
       data="char"/>
      <toRemoteFile name="text.txt">
        <partner name="UnixP_1">
          <transferAdmission>
            <ftacAdmission ftacAdmission="FTACADM1"/>
          </transferAdmission>
        </partner>
        <directory name="frg_eis_13"/>
      </toRemoteFile>
    </transferFile>
    <transferFile>
      <fromLocalFile name=
       "W:/openFT/ftscript/Test/data/large/bin.mp3"
        data="bin"/>
      <toRemoteFile name="bin.mp3">
        <partner name="WindowsP_1">
          <transferAdmission>
            <ftacAdmission ftacAdmission="FTACADM2"/>
          </transferAdmission>
        </partner>
        <directory name="frg_eis_13"/>
      </toRemoteFile>
    </transferFile>
  </parallel>
</ftscript>
```

The files *text.txt* and *bin.mp3* are delivered "simultaneously".

In fact, the file transfer requests are sent to openFT "simultaneously". If enough capacity is free then these requests are executed simultaneously.

4.19 partner

You use *partner* to specify the partner for which the activity applies or at which the file or directory is located.

If a partner is specified then this is always considered to be the remote system even if your own computer and own user ID are being addressed. If you do not specify a partner the activity refers to the current user ID on the local computer.

After de-referencing, the partner must possess a transfer admission (see section “transferAdmission” on page 86).

Format

```
<partner id="ID" ref?="ref" name="name" systemType?=
"any|unix|windows|zos|bs2000" >
  comment?
  transferAdmission
  processingAdmission?
</partner>
```

Attributes

Name	Value	Meaning
id	string	A unique ID in the current context. The context object is referenced under this ID.
ref	string	References a context object of type partner. After de-referencing, the partner must possess a transfer admission (see section “transferAdmission” on page 86).
name	string	Name of the partner system TNS and DNS names are permitted. IPv4 and IPv6 addresses start with %IP (see openFT user manual).
systemType?	any | unix | windows | zos | bs2000	The default value is *any*. Specifies the partner's system type. The system type is not determined automatically. If no system type is specified then the general data (e.g. name) is used instead of the system-specific data (e.g. unixname).

4.19.1 processingAdmission

You use *processingAdmission* to assign the processing admission for scripts.

i *processingAdmission* is not currently supported. The attribute can be specified but has no effect.

Format

```
<processingAdmission userId?="user" userAccount?="account"
userPassword?="password" >
  comment?
</processingAdmission>
```

Attributes

Name	Value	Meaning
`userId`	string	User ID Does not have to be specified if the *transferAdmission* is used (section “transferAdmission” on page 86).
`userAccount?`	string	Account information
`userPassword?`	string	Password for the user ID.

4.19.2 transferAdmission

You use *transferAdmission* to assign the admission for transfer files.

Format

```
<transferAdmission>
  comment?
  ftacAdmission | userAdmission
</transferAdmission>
```

ftacAdmission

You use *ftacAdmission* to assign the admission in the form of an FTAC transfer admission.

Format

```
<ftacAdmission ftacAdmission="ftac" />
```

Attributes

Name	Value	Meaning
`ftacAdmission`	string	FTAC transfer admission (see openFT user guide).

userAdmission

You use *userAdmission* to specify the admission in the form of the login/LOGON access data (user ID, password and account).

i Avoid using *userAdmission* in this version. openFT-Script requires the password to be specified in plain text.
If possible, use the more reliable *ftacAdmission* instead.

Format

```
<userAdmission userId="userId" userAccount?="account"
userPassword?="password" >
  comment?
</userAdmission>
```

Attributes

Name	**Value**	**Meaning**
`userId?`	string	User ID
`userAccount?`	string	Account information
`userPassword?`	string	Password for the user ID.

4.20 script

You use *script* to specify a text string which is to be executed as an operating system command. A context object of type *script* can be referenced in the *executeScript* activity or by *remoteSuccessScript* or *remoteFailureScript*.

The command must be written in the operating system-specific syntax of each operating system. You can specify a different text string for each operating system in a *script* object.

openFT-Script selects the text string that is to be executed on the basis of the operating system specification in the partner definition or on the basis of the local operating system. If the operating system is unknown or no *script* matching the operating system is specified then the text string designated with *script* is executed.

The text string (*script*) is completely output by the *ftshwact* command. If this string contains passwords or related security-relevant information (e.g. for an *ncopy* command), then the *ftshwact* command outputs this information.

A return value other than zero is interpreted as an error and results in an *ft_scriptError*.

Restrictions

1. A script (e.g. bs2000Script, unixScript) may only occupy one line and its length is limited to 500 characters.
2. In this version, the script is not configurable.

Format

```
<script id="ID" ref?="ID" repeatable?=true|false" >
  comment?
  script?
  bs2000Script?
  unixScript?
  windowsScript?
  zosScript?
  partner?
</script>
```

Attributes

Name	Value	Meaning
`id`	string	A unique ID in the current context. The context object is referenced under this ID.
`ref`	string	ID of another script object.
`repeatable?`	`true` | `false`	The default value is *true*. The script may be repeated on a restart. If *false* is specified then the script may not be repeated. If, on restart, it is not possible to determine whether the Ftscript has run then the entire Ftscript is cancelled.

Examples

See section “executeScript” on page 63.

4.21 sequence

You use *sequence* to execute the activities in a sequence one after the other in the specified order.

Each child activity can use the results of the preceding child activities. If an activity adds new context objects to the *sequence* context (e.g. *transferFile/toLocalTmpFile* or *listDirectory*), then the following activities can access the new data.

Format

```
<sequence>
  comment?
  context?
  Activity+
</sequence>
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <sequence>
    <transferFile>
      <fromLocalFile name=
       "W:/openFT/ftscript/Test/data/small/text.txt" data="char"/>
      <toRemoteFile name="text.txt">
        <partner name="UnixP_1">
          <transferAdmission>
            <ftacAdmission ftacAdmission="FTACADM1"/>
          </transferAdmission>
        </partner>
        <directory name="frg_eis_14"/>
      </toRemoteFile>
    </transferFile>
    <transferFile>
      <fromLocalFile name=
        "W:/openFT/ftscript/Test/data/large/bin.mp3" data="bin"/>
      <toRemoteFile name="bin.mp3">
        <partner name="WindowsP_1">
          <transferAdmission>
            <ftacAdmission ftacAdmission="FTACADM2"/>
          </transferAdmission>
        </partner>
        <directory name="frg_eis_14"/>
      </toRemoteFile>
    </transferFile>
  </sequence>
</ftscript>
```

The files *text.txt* and *bin.mp3* are delivered one after the other.

First of all, the first request in the request queue is submitted. Once this has been completed and if no error message is triggered, the second request in the request queue is submitted.

4.22 transferFile

You use *transferFile* to perform a file transfer. *transferFile* starts the file transfer and waits inside *ftscript* for the end of the file transfer.
The file transfer itself can be restarted.

If you specify *remoteSuccessScript* (see section “remoteSuccessScript” on page 109) or *remoteFailureScript* (see section “remoteFailureScript” on page 108) then the corresponding script is subsequently run on the remote computer.
For local scripts, you should use *faulthandlers* (see section “faulthandler” on page 68) or *executeScript* (see section “executeScript” on page 63). For an example, see page 97.

Types of file transfer

You use *transferFile* to transfer files as follows:

- File transfers from "remote" to "local" (*fromRemoteFile toLocalFile*)
- File transfers from "local" to "remote" (*fromLocalFile toRemoteFile*)
- File transfers from "remote" to "remote" can be accomplished by means of two sequential *transferFile* activities, e.g in a *sequence* activity:
  - File transfer from "remote" to "localTmp" (*fromRemoteFile toLocalTmpFile*)
  - File transfer from "localTmp" to "remote" (*fromLocalTmpFile toRemoteFile*)

Restrictions

1. After de-referencing, *fromRemoteFile* and *toRemoteFile* must possess a partner specification.
2. The elements *fromLocalFile* and *toLocalFile* must not possess any partner specification after de-referencing.
3. You cannot use *transferFile* to perform any file transfers from "local" to "local".

   i To perform file transfers from "local" to "local", you must use a corresponding script (*copy*) or specify one of the two files as "remote" and specify the local computer as the partner (specify *transferAdmission*).

Format

```
<transferFile compress?="none|byteRep|zip" writeMode?=
"replace|new|extend" transparentMode?="true|false"
dataEncryption?="yes|no|onlyDataIntegrity">
  comment?
  context?
  ( (fromRemoteFile   toLocalFile)    |
    (fromLocalFile    toRemoteFile)   |
    (fromRemoteFile   toLocalTmpFile) |
    (fromLocalTmpFile toRemoteFile)  )
  remoteSuccessScript?
  remoteFailureScript?
</transferFile>
```

Attributes

Name	Value	Meaning
`compress?`	`none` | `byteRep` | `zip`	The default value is *none*. The file is not compressed. If *byteRep* is specified then identical sequences of characters are compressed. If *zip* is specified then zip compression is used.
`writeMode?`	`replace` | `new` | `extend`	The default value is *replace*. If the file exists then it is overwritten. If the file does not exist, it is created. If *extend* is specified then the data is appended to the existing file. If the file does not exist, it is created. If *new* is specified then a new file is created. If a file with this name already exists then the activity is cancelled with the error *ft_exist*.
`transparentMode?`	`true` | `false`	The default value is *false*. The file is transferred as standard. If *true* is specified then a transparent transfer is performed (e.g. in the case of transfers from a BS2000 system to another BS2000 system via a Windows/Unix system).
`dataEncryption?`	`yes` | `no` | `onlyDataIntegrity`	The default value is *no*. The user data is not encrypted. If *yes* is specified then the user data is encrypted (for settings see the openFT user manual). If *onlyDataIntegrity* is specified then only the data integrity is checked.

Examples

1. File transfer with *remoteSuccessScript* / *remoteFailureScript*

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <script id="everything ok">
      <unixScript><![CDATA[echo everything ok
           >frg_eis_15/status.txt]]>
      </unixScript>
      <windowsScript>
         <![CDATA[cmd /c "echo everything ok"
           >frg_eis_15\status.txt]]>
      </windowsScript>
    </script>
    <script id="something failed">
      <unixScript><![CDATA[echo something failed>
           frg_eis_15/status.txt]]>
      </unixScript>
      <windowsScript>
         <![CDATA[cmd /c echo something failed
           >frg_eis_15\status.txt]]>
      </windowsScript>
    </script>
    <partner id="remote" name="UnixP_1" systemType="unix">
      <transferAdmission>
        <ftacAdmission ftacAdmission="FTACADM1"/>
      </transferAdmission>
    </partner>
  </context>
  <sequence>
    <context>
      <faulthandler>
        <default>
          <executeScript ref="something failed"/>
        </default>
      </faulthandler>
    </context>
    <transferFile>
      <fromRemoteFile name="bin.mp3">
        <partner ref="remote"/>
        <directory name="frg_eis_15"/>
      </fromRemoteFile>
      <toLocalFile name="bin.mp3">
        <directory name="frg_eis_15"/>
      </toLocalFile>
      <remoteSuccessScript ref="everything ok"/>
      <remoteFailureScript ref="something failed"/>
    </transferFile
```

Example 1 (cont.)

```
 <executeScript ref="everything ok"/>
  </sequence>
<transferFile writeMode="extend">
    <fromRemoteFile name="status.txt" data="char">
      <partner ref="remote"/>
      <directory name="frg_eis_15"/>
    </fromRemoteFile>
    <toLocalFile name="status.txt">
      <directory name="frg_eis_15"/>
    </toLocalFile>
  </transferFile>
</ftscript>
```

The file *bin.mp3* is transferred from the partner *remote* to the local file *bin.mp3*. The file *status.txt* is then created with the following contents:

– if transfer is OK

```
everything ok
everything ok
```

– if an error occurs

```
something failed
something failed
```

If errors occur during the compilation of the file *status.txt* then the script is aborted.

2. File transfer from "remote" to "remote"

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <partner id="remote1" name="UnixP_1">
      <transferAdmission>
        <ftacAdmission ftacAdmission="FTACADM1"/>
      </transferAdmission>
    </partner>
    <partner id="remote2" name="WindowsP_1">
      <transferAdmission>
        <ftacAdmission ftacAdmission="FTACADM2"/>
      </transferAdmission>
    </partner>
  </context>
  <transferFile>
    <fromRemoteFile name="data.txt">
      <partner ref="remote1"/>
      <directory name="frg_eis_16"/>
    <autoDataSpec charPattern="*.txt" binPattern="*.dat *.mp3"/>
    </fromRemoteFile>
    <toLocalTmpFile id="tmp"/>
  </transferFile>
  <transferFile>
    <fromLocalTmpFile use="tmp"/>
    <toRemoteFile name="data.txt">
      <partner ref="remote2"/>
      <directory name="frg_eis_16"/>
    </toRemoteFile>
  </transferFile>
</ftscript>
```

In the example, the file *data.txt* is first copied from the partner *remote1* to a temporary file. The temporary file is given an internal name. The suffix of the temporary file corresponds to the suffix of the associated *fromRemoteFile* (here **.txt*).

Conversion is performed using *autoDataSpec charPattern* for character symbols because the filename suffix corresponds to the pattern *.txt* (see section “autoDataSpec” on page 98). If the local system is a Windows system then the line ends are converted accordingly.

When the temporary file is transferred to the remote system *remote2*, the *autoDataSpec* settings made when the temporary file was created are taken over. They are inherited as well as *maxRecSize* and the data properties of the *fromRemoteFile* element. If the local system is a Windows system then the data is not converted on the second transfer. In the case of a local Unix system, the reverse would be true. When the transfer to the local system is performed, no data is converted. Instead, the data is converted on the subsequent transfer to the remote system.

3. Use of *faulthandler* and *executeScript* for local scripts

faulthandler (see section “faulthandler” on page 68) corresponds to the *localFailureScript*, and the *executeScript*, which directly follows *transferFile* (see section “executeScript” on page 63) corresponds to the *localSuccessScript*.

You can activate a *transferFile* request with *local*Script* as indicated in the example below:

```
...
<sequence>
  <context>
   <faulthandler>
      <default>
        <executeScript>
          .... localFailureScript...
        </executeScript>
      </default>
   </faulthandler>
  </context>
  <transferFile>
  ....
  </transferFile>
  <executeScript>
  ... localSuccessScript...
  </executeScript>
<sequence>
...
```

This *sequence* can also be located in a *parallel* or *foreach - parallel* statement.

autoDataSpec

You use *autoDataSpec* to define the transfer mode for the file in which the element was specified.

If the file's data type is unknown then it is determined on the basis of the file name using pattern recognition.
The only pattern currently permitted is **.xxx* since only the file name suffix is checked.
The file name specified for the actual transfer is used.

i If a Unix-specific file name and a Windows-specific file name are specified then the appropriate file name is analyzed depending on the partner in question.

The file types are described in the openFT user manual.

Data type	Format	Example
`char`	Text format	*.xml
`bin`	Binary format	*.doc in a Windows system
`user`	User format	

autoDataSpec describes which pattern is assigned to which data type.
If multiple patterns for different file formats match a file name then the file type is determined in the sequence *bin*, *char*, *user*.

If an explicit data type is specified for the file after de-referencing then *autoDataSpec* is not evaluated.

Restrictions

The only useful pattern is **.xxx* where *xxx* may be of any length. However, the overall length of 512 characters for the entire expression may not be exceeded.
Only the file name suffix is checked.

Format

```
<autoDataSpec ref?="ID"
        charPattern?="patternList" binPattern?="patternList"
        userPattern?="patternList" default?="char|bin|user" >
  comment?
</autoDataSpec>
```

Attributes

Name	Value	Meaning
`ref?`	string	Reference to an *autoDataSpec* context object
`binPattern?`	patternlist[1]	Wildcard pattern. If the pattern matches the specified file name then the file is transferred in *bin* format.
`charPattern?`	patternlist[1]	Wildcard pattern. If the pattern matches the specified file name then the file is transferred in *char* format.
`userPattern?`	patternlist[1]	Wildcard pattern. If the pattern matches the specified file name then the file is transferred in *user* format.
`default?`	`char` | `bin` | `user`	The default value is *char*. Specifies the assumed data type if no pattern is found.

[1] patternlist is the list of patterns separated by spaces.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <list id="FileList">
      <file name="bin.mp3"/>
      <file name="text.txt"/>
    </list>
    <list id="HostList">
      <partner name="UnixP_1">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM1"/>
        </transferAdmission>
      </partner>
      <partner name="WindowsP_1">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM2"/>
        </transferAdmission>
      </partner>
    </list>
  </context>
  <foreach listRef="HostList" selectType="partner"
           contextObject="host">
    <foreach listRef="FileList" selectType="file"
           contextObject="file">
      <transferFile>
        <fromLocalFile ref="file">
          <directory name="W:/openFT/ftscript/Test/data/small"/>
          <autoDataSpec charPattern="*.txt"
                        userPattern="*.tab *.dat" default="bin"/>
        </fromLocalFile>
        <toRemoteFile ref="file">
          <partner ref="host"/>
          <directory name="frg_eis_17"/>
        </toRemoteFile>
      </transferFile>
    </foreach>
  </foreach>
</ftscript>
```

In the example, the file list *FileList* of local files is transferred to all the computers in the *HostList*. The file transfer modes are activated in accordance with the *autoDataSpec* specification. **.txt* files are converted in accordance with the target system type. All unknown file types are transferred in binary form. **.tab* and **.dat* are defined for *user* transfers.

fromLocalFile

You use *fromLocalFile* to specify the local source file for file transfer.

The transfer type (data type) can be derived from the file name by means of *autoDataSpec* (see section “autoDataSpec” on page 98).

File names may be a maximum of 512 characters in length. This is checked before the Ftscript is run. The length of the resulting file name (*directory* and *filenames*) is limited by the operating system in question and the openFT version. The length is the number of characters plus 1 character if *directory* does not end with a "/". The check is performed while the Ftscript is being run.
For further information, see section “File name attributes” on page 20.

Restrictions

fromLocalFile must not contain any partner specification after de-referencing.

Format

```
<fromLocalFile ref?="ID" data?="auto|char|bin|user"
               recordFormat?="std|undef|var|fix"
               maxRecSize?="int" ccsname?="string" filenames >
  comment?
  directory?
  autoDataSpec?
</fromLocalFile
```

Attributes

Name	**Value**	**Meaning**
`ref?`	string	Reference to another file object *partner* and *directory* are taken over from this if you have not specified the elements here (see section “Referencing” on page 15).
`data?`	`auto`|`char`|`bin`|`user`	The default value is *auto*. Specifies the data type. If *char*, *bin* or *user* are specified then the *autoDataSpec* specification is ignored (see section “autoDataSpec” on page 98).

Name	Value	Meaning
`record-Format?`	`std` | `undef` | `var` | `fix`	The default value is *std*. Specifies the record format. The standard openFT assignment applies (*data=bin* -> *undef*, otherwise *var*) If *undef* is specified then the record format is undefined, e.g. in the case of binary formats. If *var* is specified then the record format is variable, e.g. in the case of text formats (1 record 1 line; lines can be of different lengths). If *fix* is specified then the record format is fixed, e.g. f80 (with *recordFormat=fix* and *maxRecSize=80*.
`maxRecSize?`	1-65535 (openFT ≤ V11: 1-32756 or 1-32767)	Specifies the record size. By default, the openFT value applies (see openFT user manual). If *data=char* then *maxRecSize* specifies the length of a line (to the CR/LF). Files with a line length greater than 65535 bytes must be transferred with *data=bin*. For example, an Ftscript with a *maxRecSize* value that is not permitted in openFT V11.0 will not run in openFT V11.0..
`ccsname?`	string max. 8 characters	Specifies the Coded Character Set (see openFT user manual).
`filenames`	See section “File name attributes” on page 20.	

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <transferFile>
    <fromLocalFile name="bin.mp3">
      <directory name="W:/openFT/ftscript/Test/data/small"/>
      <autoDataSpec charPattern="*.txt" userPattern="*.tab *.dat"
           default="bin"/>
    </fromLocalFile>
    <toRemoteFile name="bin.mp3">
      <partner name="UnixP_1">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM1"/>
        </transferAdmission>
      </partner>
      <directory name="frg_eis_18"/>
    </toRemoteFile>
  </transferFile>
</ftscript>
```

In the example, the local file *bin.mp3* is transferred to the remote system *UnixP_1* under the name *bin.mp3*.
No transfer mode or *autoDataSpec* is specified. The transfer mode is *bin* (default value of *autoDataSpec*).

fromLocalTmpFile

You use *fromLocalTmpFile* to specify a temporary source file for file transfer from "remote" to "remote" (see “Types of file transfer” on page 92).

The temporary file is simply a buffer under the local ID. This temporary file is deleted automatically.

The *data*, *maxRecSize*, *recordFormat*, *ccsname* and *autoDataSpec* specifications in the *fromRemoteFile toLocalTmpFile* activity apply implicitly. When a *TmpFile* is created from *fromRemoteFile* then these specifications are inherited.

Restrictions

You may only use *fromLocalTmpFile* after a *fromRemoteFile toLocalTmpFile* activity since the file *TmpFile* already exists and must be accessible in the current context. For more information, see section “fromRemoteFile” on page 106 and section “toLocalTmpFile” on page 111.

Format

```
<fromLocalTmpFile use="tmpID"/>
```

Attributes

Name	Value	Meaning
use	string	Reference to a file *TmpFile* which can be accessed in the current context. This file must previously have been created with a *transferFile* using *toLocalTmpFile*.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ftscript version="1">
  <context>
    <partner id="FileServer" name="UnixP_1">
      <transferAdmission>
        <ftacAdmission ftacAdmission="FTACADM1"/>
      </transferAdmission>
    </partner>
    <list id="RemoteHostList">
      <partner name="UnixP_1">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM1"/>
        </transferAdmission>
      </partner>
      <partner name="WindowsP_1">
        <transferAdmission>
          <ftacAdmission ftacAdmission="FTACADM2"/>
        </transferAdmission>
      </partner>
    </list>
  </context>
  <listDirectory listObject="FileList">
    <partner ref="FileServer"/>
    <baseDir name="frg_eis_19/datastore"/>
  </listDirectory>
  <foreach listRef="FileList" selectType="file"
          contextObject="File" execute="parallel">
    <transferFile>
      <fromRemoteFile ref="File">
        <partner ref="FileServer"/>
        <directory name="frg_eis_19/datastore"/>
      </fromRemoteFile>
      <toLocalTmpFile id="tmpFile"/>
    </transferFile>
    <foreach listRef="RemoteHostList" selectType="partner"
            contextObject="RemoteHost" execute="parallel">
      <transferFile>
        <fromLocalTmpFile use="tmpFile"/>
        <toRemoteFile ref="File">
          <partner ref="RemoteHost"/>
          <directory name="frg_eis_19/targetDir"/>
        </toRemoteFile>
      </transferFile>
    </foreach>
  </foreach>
</ftscript>
```

In the example, the files in the directory *frg_eis_19/datastore* are copied to the relevant *frg_eis_19/targetDir* target directory on the various computers.

listDirectory is used to identify the files on the *FileServer*.

The first *foreach* activity works through all the files identified on the *FileServer*. Each file is copied to a temporary file *tmpFile*.
In the second *foreach* activity, each temporary file is copied to the target directory *frg_eis_19/targetDir* on the *RemoteHost*.

Finally, the associated temporary file *tmpFile* is deleted.

fromRemoteFile

You use *fromRemoteFile* to specify the remote source file for file transfer.

The transfer type (data type) can be derived from the file name by means of *autoDataSpec* (see section “autoDataSpec” on page 98).

File names may be a maximum of 512 characters in length. This is checked before the Ftscript is run. The length of the resulting file name (*directory* and *filenames*) is limited by the operating system in question and the openFT version. The length is the number of characters plus 1 character if *directory* does not end with a "/". The check is performed while the Ftscript is being run.
For further information, see section “File name attributes” on page 20.

Restrictions

fromRemoteFile must contain a partner specification after de-referencing.

Format

```
<fromRemoteFile ref?="ID" data?="auto|char|bin|user"
                recordFormat?="std|undef|var|fix"
                maxRecSize?="int" ccsname?="string" filenames
  comment?
  partner?
  directory?
  autoDataSpec?
</fromRemoteFile
```

Attributes

Name	Value	Meaning
`ref?`	string	Reference to another file object *partner* and *directory* are taken over from this if you have not specified the elements here (see section “Referencing” on page 15).
`data?`	`auto` | `char` | `bin` | `user`	The default value is *auto*. Specifies the data type. If *char*, *bin* or *user* are specified then the *AutoDataSpec* specification is ignored (see section “autoDataSpec” on page 98).
`record-Format?`	`std` | `undef` | `var` | `fix`	The default value is *std*. Specifies the record format. The standard openFT assignment applies (*data=bin* -> *undef*, otherwise *var*) If *undef* is specified then the record format is undefined, e.g. in the case of binary formats. If *var* is specified then the record format is variable, e.g. in the case of text formats (1 record 1 line; lines can be of different lengths). If *fix* is specified then the record format is fixed, e.g. f80 (with *recordFormat=fix* and *maxRecSize=80*.
`maxRecSize?`	1-65535 (openFT ≤ V11: 1-32756 or 1-32767)	Specifies the record size. By default, the openFT value applies (see openFT user manual). If *data=char* then *maxRecSize* specifies the length of a line (to the CR/LF). Files with a line length greater than 65535 bytes must be transferred with *data=bin*. For example, an Ftscript with a *maxRecSize* value that is not permitted in openFT V11.0 will not run in openFT V11.0
`ccsname?`	string max. 8 characters	Specifies the Coded Character Set (see openFT user manual).
`filenames`	See section “File name attributes” on page 20.	

Example

See example on page 94.

remoteFailureScript

You use *remoteFailureScript* to execute a script on the remote system if data transfer fails (see section “script” on page 88).
The functionality *-rf* of the openFT request is used (see *ft* command in the openFT User Guide).

i You should note that *remoteFailureScript* and a corresponding *faulthandler* in the Ftscript (see section “faulthandler” on page 68) may affect one another since their execution is practically simultaneous (local and remote).

See also section “executeScript” on page 63.

Additional notes

The script is also executed if a running data transfer is aborted with a *fault* (internally, *ft_cancel* is issued if a *fault* occurs, see section “fault” on page 65 and the openFT user manual).

If a *fault* occurs then *remoteFailureScript* is only executed for the running transfers.

Format

```
<remoteFailureScript ref?="ID">
  comment?
  script?
  bs2000Script?
  unixScript?
  windowsScript?
  zosScript?
</remoteFailureScript>
```

Attributes

Name	Value	Meaning
`ref?`	string	Reference to a script context object. Any partner described there is not evaluated. The partner in the associated *transferFile* activity is used. The specification of *repeatable* in referenced script objects is ignored.

Example

See example on page 94.

remoteSuccessScript

You use *remoteSuccessScript* to execute a script on the remote system if data transfer succeeds (see section “script” on page 88).
The functionality *-rs* of the openFT request is used (see *ft* command in the openFT user manual).

See also section “executeScript” on page 63.

Format

```
<remoteSuccessScript ref?="ID">
  comment?
  script?
  bs2000Script?
  unixScript?
  windowsScript?
  zosScript?
</remoteSuccessScript>
```

Attributes

Name	Value	Meaning
`ref?`	string	Reference to a script context object. Any partner described there is not evaluated. The partner in the associated *transferFile* activity is used. The specification of *repeatable* in referenced script objects is ignored.

Example

See example on page 94.

toLocalFile

You use *toLocalFile* to specify the local target file for file transfer.

See also section “file” on page 72.

File names may be a maximum of 512 characters in length. This is checked before the Ftscript is run. The length of the resulting file name (*directory* and *filenames*) is limited by the operating system in question and the openFT version. The length is the number of characters plus 1 character if *directory* does not end with a "/". The check is performed while the Ftscript is being run.
For further information, see section “File name attributes” on page 20.

Restrictions

toLocalFile must not contain any partner specification after de-referencing.

Format

```
<toLocalFile ref?="ID" ccsname?="string" filenames>
  comment?
  partner?
  directory?
</toLocalFile>
```

Attributes

Name	Value	Meaning
`ref?`	string	Reference to another file object
`ccsname?`	string max. 8 characters	Specifies the Coded Character Set (see openFT user manual).
`filenames`	See section “File name attributes” on page 20.	

Example

See example on page 94.

toLocalTmpFile

You use *toLocalTmpFile* to specify a temporary target file for file transfer from "remote" to "remote" (see “Types of file transfer” on page 92).

The temporary file is simply a buffer under the local ID. The file is assigned to a file object with the specified ID. This file object is displayed in the parent context of the current *transferFile* element and must not already exist there.

The file is given an internal name. The suffix of the file is determined from the suffix of the associated *fromRemoteFile*. If this does not have a suffix then the generated temporary file also has no suffix. This file is stored in user memory. If the memory space for the user is restricted (Disk Quota) then the limit may be exceeded when temporary files are created. The *transferFile* activity is cancelled with the error code *ft_err_LOCERR_MEM*.

The file object inherits the *data*, *MaxRecSize*, *recordFormat*, *ccsname* and *autoDataSpec* specifications in the associated *fromRemoteFile* activity. These are then re-used when the *fromLocalTmpFile* activity is called (see section “fromLocalTmpFile” on page 103).

The temporary file is deleted automatically as soon as the context in which the file object was defined is exited.

See also section “file” on page 72.

Restrictions

1. You can only access the generated temporary file with *use="tmpID"* from a *fromLocalTmpFile*.
2. The *transferFile fromRemoteFile* activity must be concluded before you use *transferFile toRemoteFile* to access the temporary file. You should therefore perform these activities sequentially or, if appropriate, embed them in a *<sequence>*.

Format

```
<toLocalTmpFile id="tmpID"/>
```

Attributes

Name	Value	Meaning
`id`	string	ID of the temporary file in *transferFile* elements associated with the parent context. You can only access this ID with *use="tmpID"* from a *fromLocalTmpFile*.

Example

See example in section “fromLocalTmpFile” on page 104.

toRemoteFile

You use *toRemoteFile* to specify the remote target file for file transfer.

See also section “file” on page 72.

File names may be a maximum of 512 characters in length. This is checked before the Ftscript is run. The length of the resulting file name (*directory* and *filenames*) is limited by the operating system in question and the openFT version. The length is the number of characters plus 1 character if *directory* does not end with a "/". The check is performed while the Ftscript is being run.
For further information, see section “File name attributes” on page 20.

Restrictions

toRemoteFile must contain a partner specification after de-referencing.

Format

```
<toRemoteFile ref?="ID" ccsname?="string" filenames>
  comment?
  partner?
  directory?
</toRemoteFile>
```

Attributes

Name	Value	Meaning
`ref?`	string	Reference to another file object
`ccsname?`	string max. 8 characters	Specifies the Coded Character Set (see openFT user manual).
`filenames`	See section “File name attributes” on page 20.	

Example

See example on page 94.

5 Error messages

For a list and description of the “severe” Ftscript error codes, please refer to section “"Severe" Ftscript error codes” on page 24.

You can use the error codes listed here in the *case* statement of the *faulthandler* activity. If required, you can find the internal code in the *ActivityObject* output parameter of the output from *ftshwact*.

The internal code has the structure *ft_codexxxx*, where *xxxx* is the openFT error number.

The Ftscript error code is assigned to the openFT error numbers (see also the openFT User Manual, *fthelp* command).

Internal code	Error code	Description
ft_activeDirNotExists	ft_resource	Script request directory does not exist. Restart not possible.
ft_alarmException	ft_error	Error during alarm handling. Is ignored.
ft_alarmFailed	ft_error	Slow poll: An unhandled error has occurred.
ft_callFtRuntime	ft_error	Runtime error on FT call.
ft_callSecurity	ft_access	Access error during file deletion.
ft_cancelCmdError	ft_error	Abort terminated with unknown error.
ft_cancelError	ft_error	Error during abort.
ft_cancelUnexpectedState	ft_error	Request terminated but status incorrect. Abort expected.
ft_cantCreateJobListener	ft_resource	Interruption during communication with openFT.
ft_cantInitializeJob	ft_error	Request cannot be initialized.
ft_cantWriteLogData	ft_resource	Log data could not be written.
ft_circleRef	ft_reference	Circular reference not permitted.
ft_closeOrderQueue	ft_resource	Order queue could not be closed correctly.
ft_code1038	ft_ignore	Request <request ID> is being completed and can no longer be deleted.

Internal code	Error code	Description
ft_code108	ft_connection	Request <request ID>. Remote system not accessible.
ft_code20	ft_notExist	<local file> not found.
ft_code2014	ft_paramError	No modification of file attributes demanded.
ft_code2015	ft_admin	openFT is not authorized to process requests for this user.
ft_code2016	ft_notEmpty	The directory <local file> is not empty.
ft_code2017	ft_localFile Structure	The file attributes do not correspond to the request parameters.
ft_code2018	ft_error	Attributes could not be modified.
ft_code2019	ft_cantCreate	<local file> could not be created.
ft_code2021	ft_notExist	CCS name not known.
ft_code2022	ft_notExist	Higher-level directory not found.
ft_code2023	ft_exist	<local file> already exists.
ft_code2024	ft_notSupported	Transfer of file generation groups not supported.
ft_code2025	ft_access	Error accessing <local file>.
ft_code2026	ft_paramError	Resulting file name <local file> too long
ft_code2027	ft_paramError	No file or directory name specified.
ft_code2028	ft_auth	Invalid management password.
ft_code2029	ft_access	<local file> not available.
ft_code2030	ft_notExist	Home directory not found.
ft_code2031	ft_access	Not possible to rename.
ft_code2032	ft_resource	Not enough storage space for <local file>.
ft_code2033	ft_notExist	File owner unknown.
ft_code2034	ft_auth	Invalid file password.
ft_code2036	ft_cantDelete	The file's retention period has not yet expired.
ft_code2037	ft_auth	<local file> is write-protected.
ft_code2038	ft_localFile Structure	File structure not supported.
ft_code2039	ft_syntax	Syntax error in resulting file name <local file>.
ft_code2040	ft_notSupported	Transparent file transfer not supported.

Internal code	Error code	Description
ft_code2042	ft_notSupported	Not possible to extend the file during transparent transfer.
ft_code2043	ft_auth	Access to <local file> prohibited
ft_code2044	ft_paramTooLong	Follow-up processing too long.
ft_code2045	ft_auth	Authorization for follow-up processing invalid.
ft_code2046	ft_auth	Local transfer admission invalid.
ft_code2047	ft_auth	Request rejected by local FTAC.
ft_code2048	ft_notSupported	Function for protocol <partner protocol type> not supported.
ft_code2049	ft_notSupported	Remote follow-up processing not supported.
ft_code2070	ft_admin	Request <request ID> openFT is no longer authorized to process requests for this user.
ft_code2071	ft_notSupported	Request <request ID>. User data encryption not installed.
ft_code2072	ft_abort	Request <request ID> has been deleted.
ft_code2073	ft_corrupt	Request <request ID>. Error during encryption.
ft_code2074	ft_cantCreate	Request <request ID>. <local file> could not be created.
ft_code2075	ft_notExist	Request <request ID>. Higher-level directory no longer found.
ft_code2076	ft_access	Request <request ID>. Error on <local file> input/output
ft_code2077	ft_access	Request <request ID>. File now locked against concurrent access.
ft_code2078	ft_access	Request <request ID>. <local file> no longer available.
ft_code2079	ft_notExist	Request <request ID>. <local file> no longer found.
ft_code2080	ft_notExist	Request <request ID>. Home directory no longer found.
ft_code2081	ft_resource	Request <request ID>. <local file> can no longer be assigned any space.
ft_code2082	ft_notExist	Request <request ID>. File owner no longer known.

Internal code	Error code	Description
ft_code2083	ft_error	Request <request ID>. Error during preprocessing/postprocessing.
ft_code2084	ft_error	Request <request ID>. Exit code <2> during preprocessing/postprocessing.
ft_code2085	ft_auth	Request <request ID>. File password no longer valid.
ft_code2086	ft_auth	Request <request ID>. <local file> is now write-protected.
ft_code2087	ft_localFile Structure	Request <request ID>. File structure error.
ft_code2088	ft_error	Request <request ID>. NDMS error <2>.
ft_code2089	ft_recoveryFailed	Request <request ID>. Recovery failed.
ft_code2090	ft_error	Request <request ID>. Error in completing file transfer.
ft_code2092	ft_auth	Request <request ID>. Access to <local file> is no longer permitted.
ft_code2093	ft_error	Request <request ID>. FTAM error <2>.
ft_code2094	ft_cantDelete	Request <request ID>. The file's retention period has not yet expired.
ft_code2095	ft_notSupported	Request <request ID>. Not possible to extend the file during transparent transfer.
ft_code2096	ft_notSupported	Request <request ID>. File structure not supported.
ft_code2109	ft_connection	Request <request ID>. Connection request rejected by local transport system.
ft_code2110	ft_connection	Request <request ID>. Data integrity checking has detected an error.
ft_code2111	ft_connection	Encryption/data integrity check not possible. Encryption is disabled.
ft_code2112	ft_connection	Request <request ID>. Data integrity check is not supported by partner.

Internal code	Error code	Description
ft_code2113	ft_connection	Request <request ID>. User data encryption not possible for this request.
ft_code2114	ft_connection	Request <request ID>. Local system ID rejected by remote system ('<partner>').
ft_code2115	ft_connection	Request <request ID>. Request interrupted by remote system.
ft_code2116	ft_connection	Local application '<1>' not defined.
ft_code2117	ft_connection	Local application '<1>' not available.
ft_code2118	ft_connection	Request <request ID>. Authentication of local system failed.
ft_code2119	ft_connection	Request <request ID>. Local system unknown in remote system.
ft_code2120	ft_connection	Remote system '<partner>' unknown.
ft_code2121	ft_connection	Request <request ID>. Authentication of partner failed.
ft_code2122	ft_connection	Request <request ID>. Connection rejected or disconnected. Cause <2>
ft_code2123	ft_connection	Request <request ID>. Error <2> on OSS call.
ft_code2124	ft_connection	Request <request ID>. No free transport connection.
ft_code2125	ft_connection	Request <request ID>. Transport connection lost.
ft_code2126	ft_connection	Request <request ID>. Transport system error. Error code <2>.
ft_code2127	ft_connection	Request <request ID>. No data traffic within <2> seconds.
ft_code2140	ft_admin	Request <request ID>. Remote system: openFT is not authorized to process requests for this user.
ft_code2141	ft_notEmpty	Request <request ID>. Remote system: The directory ’<remote file>’ is not empty.
ft_code2142	ft_remoteFile Structure	Request <request ID>. Remote system: The file attributes do not correspond to the request parameters.

Internal code	Error code	Description
ft_code2143	ft_access	Request <request ID>. Remote system: Attributes could not be modified.
ft_code2144	ft_cantCreate	Request <request ID>. Remote system: File/directory ’<remote file>’ could not be created.
ft_code2145	ft_notExist	Request <request ID>. Remote system: CCS name unknown or not supported.
ft_code2146	ft_notExist	Request <request ID>. Remote system: Higher-level directory not found.
ft_code2147	ft_exist	Request <request ID>. Remote system: File/directory ’<remote file>’ already exists.
ft_code2148	ft_notSupported	Request <request ID>. Remote system: Transfer of file generation groups not supported.
ft_code2149	ft_access	Request <request ID>. Remote system: Error accessing ’<remote file>'.
ft_code2150	ft_syntax	Request <request ID>. Remote system: Resulting file name too long.
ft_code2151	ft_access	Request <request ID>. Remote system: File locked against concurrent access.
ft_code2152	ft_paramError	Request <request ID>. Remote system: No file or directory name specified.
ft_code2153	ft_auth	Request <request ID>. Remote system: Invalid management password.
ft_code2154	ft_access	Request <request ID>. Remote system: File/directory '<remote file>' not available
ft_code2155	ft_notExist	Request <request ID>. Remote system: File/directory '<remote file>' not found.
ft_code2156	ft_notExist	Request <request ID>. Remote system: Home directory not found.

Internal code	Error code	Description
ft_code2157	ft_access	Request <request ID>. Remote system: Not possible to rename.
ft_code2158	ft_resource	Request <request ID>. Remote system: Not enough storage space for '<remote file>'.
ft_code2159	ft_notExist	Request <request ID>. Remote system: File owner unknown.
ft_code2160	ft_auth	Request <request ID>. Remote system: Invalid file password.
ft_code2161	ft_cantDelete	Request <request ID>. Remote system: The file's retention period has not yet expired.
ft_code2162	ft_auth	Request <request ID>. Remote system: File/directory '<remote file>' is write-protected.
ft_code2163	ft_remoteFile Structure	Request <request ID>. Remote system: File structure not supported.
ft_code2164	ft_syntax	Request <request ID>. Remote system: Syntax error in resulting file name.
ft_code2165	ft_notSupported	Request <request ID>. Remote system: Transparent file transfer not supported.
ft_code2166	ft_notSupported	Request <request ID>. Remote system: Not possible to extend the file during transparent transfer.
ft_code2167	ft_auth	Request <request ID>. Remote system: Access to '<remote file>' prohibited
ft_code2168	ft_paramTooLong	Request <request ID>. Remote system: Follow-up processing too long.
ft_code2169	ft_auth	Request <request ID>. Remote system: Transfer admission invalid.
ft_code2170	ft_notSupported	Request <request ID>. Remote system: Function not supported.
ft_code2195	ft_admin	Request <request ID>. Remote system: openFT is no longer authorized to process requests for this user.

Internal code	Error code	Description
ft_code2196	ft_abort	Request <request ID> has been deleted in the remote system.
ft_code2197	ft_cantCreate	Request <request ID>. Remote system: File/directory ’<remote file>’ could not be created.
ft_code2198	ft_notExist	Request <request ID>. Remote system: Higher-level directory no longer found.
ft_code2199	ft_access	Request <request ID>. Remote system: Error on '<remote file>' input/output
ft_code2200	ft_access	Request <request ID>. Remote system: File now locked against concurrent access.
ft_code2201	ft_access	Request <request ID>. Remote system: File/directory '<remote file>' no longer available.
ft_code2202	ft_notExist	Request <request ID>. Remote system: File/directory '<remote file>' no longer found.
ft_code2203	ft_notExist	Request <request ID>. Remote system: Home directory no longer found.
ft_code2204	ft_resource	Request <request ID>. Remote system: File/directory '<remote file>' can no longer be assigned any space.
ft_code2205	ft_notExist	Request <request ID>. Remote system: File owner no longer known.
ft_code2206	ft_error	Request <request ID>. Remote system: Error during preprocessing/postprocessing.
ft_code2207	ft_error	Request <request ID>. Remote system: Exit code <2> during preprocessing/postprocessing.
ft_code2208	ft_auth	Request <request ID>. Remote system: File password no longer valid.
ft_code2209	ft_auth	Request <request ID>. Remote system: File/directory '<remote file>' is now write-protected.

Internal code	Error code	Description
ft_code2210	ft_remoteFile Structure	Request <request ID>. Remote system: File structure error.
ft_code2211	ft_error	Request <request ID>. Remote system: NDMS error <2>.
ft_code2212	ft_recoveryFailed	Request <request ID>. Remote system: Recovery failed.
ft_code2213	ft_resource	Request <request ID>. Remote system: Resource bottleneck.
ft_code2214	ft_auth	Request <request ID>. Remote system: Access to '<remote file>' is no longer permitted.
ft_code2215	ft_error	Request <request ID>. FTAM error <2>.
ft_code2216	ft_remoteFile Structure	Request <request ID>. Remote system: File structure not supported.
ft_code2217	ft_cantDelete	Request <request ID>. Remote system: The file's retention period has not yet expired.
ft_code2218	ft_notSupported	Request <request ID>. Remote system: Not possible to extend the file during transparent transfer.
ft_code2226	ft_error	Inconsistent monitor file contents.
ft_code2227	ft_error	Monitor file not used by openFT.
ft_code2228	ft_error	Monitor file not present.
ft_code236	ft_admin	Set instance '<1>' no longer found.
ft_code35	ft_access	File locked against concurrent access.
ft_code41	ft_resource	Request queue full.
ft_code700	ft_syntax	The parameters '<1>' and '<2>' may not both be specified at once.
ft_code701	ft_syntax	Input error.
ft_code702	ft_syntax	Parameter value '<1>' too long.
ft_code703	ft_syntax	Mandatory parameter missing.
ft_code704	ft_syntax	Mandatory parameter '<1>' missing.
ft_code705	ft_syntax	Parameter '<1>' specified more than once.
ft_code706	ft_syntax	Parameter '<1>' can only be specified together with '<2>'.
ft_code707	ft_syntax	Invalid parameter '<1>'.

Internal code	Error code	Description
ft_code708	ft_syntax	Range of values for parameter '<1>' not respected.
ft_code709	ft_syntax	Too many positional parameters.
ft_code710	ft_syntax	Incorrect parameter value '<1>'.
ft_code750	ft_syntax	Command not known.
ft_code751	ft_syntax	Command name ambiguous with regard to '<1>'.
ft_code752	ft_syntax	Closing bracket missing for operand '<1>'.
ft_code753	ft_syntax	Incorrect separator '<1>' after operand '<2>'.
ft_code755	ft_syntax	List value of operand '<1>' not compatible with data type '<2>'.
ft_code756	ft_syntax	Introductory operand value required for '<1>'.
ft_code757	ft_syntax	Value of operand '<1>' not compatible with data type '<2>'.
ft_code758	ft_syntax	Keyword value of operand '<1>' ambiguous with regard to '<2>'.
ft_code759	ft_syntax	Too many closing brackets.
ft_code760	ft_syntax	Required operand '<1>' not present.
ft_code762	ft_syntax	Operand name '<1>' ambiguous with regard to '<2>'.
ft_code763	ft_syntax	Operand '<1>' not known.
ft_code764	ft_syntax	Operand '<1>' specified more than once.
ft_code765	ft_syntax	Too many list elements for operand '<1>'.
ft_code766	ft_syntax	Too many positional operands.
ft_code767	ft_syntax	Too many positional operands for '<1>'.
ft_code780	ft_syntax	Internal error: Insufficient operand memory.
ft_code781	ft_syntax	Internal error: Structure nesting too deep.
ft_code790	ft_syntax	Available commands: '<1>'.
ft_code791	ft_syntax	Available list values: '<1>'.
ft_code792	ft_syntax	Available operands: '<1>'.
ft_code793	ft_syntax	Available values: '<1>'.
ft_code800	ft_error	Request <request ID>. Internal error. Monitor file not accessible.
ft_code801	ft_error	Request <request ID>. Internal error.

Internal code	Error code	Description
ft_code802	ft_error	Request <request ID>. Warning: Inconsistent monitor file '<2>' contents.
ft_code803	ft_error	Request <request ID>. Follow-up processing could not be started.
ft_code804	ft_error	Request <request ID>. Inconsistent request data.
ft_code850	ft_error	Internal error. Monitor file not accessible.
ft_code851	ft_error	Internal error.
ft_code852	ft_error	Internal error. Set instance '<1>' incom-patible.
ft_code853	ft_error	Reloading error. Error code <1>.
ft_code854	ft_error	No longer possible to write logging records. Process terminated.
ft_code855	ft_error	No further storage space for internal files.
ft_code856	ft_error	Error on OPS output.
ft_code857	ft_error	Error in key file '<1>'.
ft_code858	ft_error	Internal error. Not possible to set/reset file locks.
ft_code859	ft_error	Function not supported due to teleservice restrictions.
ft_code860	ft_error	Teleservice restriction for FTAC due to FT.
ft_code861	ft_error	Teleservice restriction for <1>.
ft_code862	ft_error	Protocol stack '<1>' not installed.
ft_code999	ft_error	openFT panic <1>. Abnormal termination.
ft_compressModeIllegal	ft_error	Defective compress mode. Defective schema?
ft_compressModeOutOfRange	ft_error	Incorrect specification in compress mode. ftAPI version?
ft_contextCantCreate	ft_error	Context file could not be created.
ft_contextCantCreate2	ft_error	Context file could not be generated.
ft_createDirectoryFailed	ft_resource	Directory could not be created or not a directory.
ft_createDirectorySecurity	ft_access	Directory could not be created.
ft_createNoFilename	ft_paramError	No name specified on generation of a directory.

Internal code	Error code	Description
ft_createParamError	ft_error	Parameter error on creation of a directory.
ft_deleteFailed	ft_error	Local file/local directory could not be deleted.
ft_deleteNoFilename	ft_paramError	No directory or file name was specified for deletion.
ft_deleteOrderQueue	ft_resource	Order queue could not be deleted.
ft_deleteParamError	ft_error	Parameter error on deletion.
ft_deleteParents	ft_error	..' Directory will not be deleted.
ft_deleteSelf	ft_error	.' Directory will not be deleted.
ft_delparError	ft_error	Error in the internal *delete* parameters.
ft_directionModeOutOfRange	ft_error	Incorrect specification for direction. ftAPI version?
ft_directoryElementUnknown	ft_error	Schema and Ftscript do not match. There is no such directory element.
ft_doubleContextObject	ft_error	Context object initialized more than once
ft_doubleLocalTmpFileRef	ft_reference	ID for local temporary file already assigned in this context.
ft_emptyEventQueue	ft_error	Internal error: Empty queue.
ft_emptyParallelQueue	ft_error	Internal error: Empty queue.
ft_emptyTransferAdmission	ft_paramError	The transfer admission does not contain any user or FTAC admission.
ft_encryptionIllegal	ft_error	Incorrect specification for encryption. Defective schema?
ft_encryptionOutOfRange	ft_error	Incorrect specification for encryption. ftAPI version?
ft_err_CONNERR_NOCONN	ft_connection	No free transport connection.
ft_err_CONNERR_NOTAVAIL	ft_connection	The remote system is not accessible.
ft_err_CONNERR_UNKNOWN	ft_configuration	The remote system is unknown.
ft_err_INT_CRFILE	ft_error	Error during file generation.
ft_err_INT_FORK_<errno>	ft_error	Error on fork system call. errno is the value of the errno variable. This value is set by the defective system call.
ft_err_INT_INIT	ft_error	It is not possible to initialize the server.
ft_err_INT_INTERNAL_FN	ft_panic	Other internal error: Function not supported.
ft_err_INT_INTERNAL_VERS	ft_panic	Other internal error: Data structure version not supported.

Internal code	Error code	Description
ft_err_INT_MEM	ft_resource	Error on storage request.
ft_err_INT_OPEN_<errno>	ft_error	Error on open system call. errno is the value of the errno variable. This value is set by the defective system call.
ft_err_INT_OPENDIR_<errno>o	ft_error	Error on opendir system call. errno is the value of the errno variable. This value is set by the defective system call.
ft_err_INT_PIPE_<errno>	ft_error	Error on pipe system call. errno is the value of the errno variable. This value is set by the defective system call.
ft_err_INT_READ_<errno>	ft_error	Error on read system call. errno is the value of the errno variable. This value is set by the defective system call.
ft_err_INT_RMFILE_<errno>	ft_error	Error on rmfile system call. errno is the value of the errno variable. This value is set by the defective system call.
ft_err_INT_SIGNAL_signal	ft_error	The command was interrupted by signal. signal indicates the signal which caused the interruption.
ft_err_INT_STAT_<errno>	ft_error	Error on stat system call. errno is the value of the errno variable. This value is set by the defective system call.
ft_err_INT_SYSTEM_<errno>	ft_error	Error on system system call. errno is the value of the errno variable. This value is set by the defective system call.
ft_err_INT_WRITE_<errno>	ft_error	Error on write system call. errno is the value of the errno variable. This value is set by the defective system call. If it was not possible to write all the bytes, errno has the value -1.
ft_err_LOCERR_EXIST	ft_exist	The local file already exists.
ft_err_LOCERR_FTAC	ft_access	The request was rejected by the local FTAC.
ft_err_LOCERR_FTC_ <exit-status>	ft_panic	<exit-status> indicates the message number of the ftc command (see message 94 and message 95 in the openFT message table).
ft_err_LOCERR_INCONS	ft_localFile Structure	The local file is inconsistent.
ft_err_LOCERR_MEM	ft_resource	The local file is assigned no space.
ft_err_LOCERR_NOACCESS	ft_access	It is not possible to access the local file.
ft_err_LOCERR_NOCREAT	ft_cantCreate	The local file cannot be created.

Internal code	Error code	Description
ft_err_LOCERR_NOTEXIST	ft_notExist	The local file cannot be found.
ft_err_PAR_DIRAC_<errno>	ft_error	errno is the value of the errno variable. This value is set by the stat() call. The errno variable has the value 0 if no write authorization has been assigned for the directory.
ft_err_PAR_FTMSG_<code>	Error code in accordance with ft_code<code>	<Message in accordance with ft_code<code>>.
ft_err_PAR_INVSESS	ft_panic	The session number is invalid.
ft_err_PAR_LEN	ft_notSupported	The name of the working directory (workdir) is too long.
ft_err_PAR_LEN_ACCOUNT	ft_paramTooLong	Parameter too long. ftamext->account.
ft_err_PAR_LEN_CRPWD	ft_paramTooLong	Parameter too long. ftamext->crpasswd.
ft_err_PAR_LEN_FPWD	ft_paramTooLong	Parameter too long. mgmtpasswd or filepasswd.
ft_err_PAR_LEN_LEGALQ	ft_paramTooLong	Parameter too long. ftamext -> legalq.
ft_err_PAR_LEN_LOCFN	ft_paramTooLong	Parameter too long. locfn.
ft_err_PAR_LEN_LOCPR	ft_paramTooLong	Parameter too long. Total of the lengths of locsuccproc and locfailproc.
ft_err_PAR_LEN_REMACC	ft_paramTooLong	Parameter too long. remaccount
ft_err_PAR_LEN_REMADM	ft_paramTooLong	Parameter too long. remadmis.
ft_err_PAR_LEN_REMFN	ft_paramTooLong	Parameter too long. fn or remfn.
ft_err_PAR_LEN_REMPR	ft_paramTooLong	Parameter too long. Total of the lengths of remsuccproc and remfailproc.
ft_err_PAR_LEN_REMPWD	ft_paramTooLong	Parameter too long. rempasswd.
ft_err_PAR_LEN_REMSYS	ft_paramTooLong	Parameter too long. remsys.
ft_err_PAR_MAND	ft_panic	– The parameter list par was not specified. – The name of the working directory (workdir) was not specified. – The output area stat was not specified. The parameter list par was not specified. – The output area info was not specified (only in the case of ft_show()).
ft_err_PAR_MAND_LOCFN	ft_paramError	locfn was not specified.

Internal code	Error code	Description
ft_err_PAR_MAND_REMSYS	ft_error	The remote system was not specified.
ft_err_PAR_NODIR	ft_panic	The specified name (workdir) does not designate a directory.
ft_err_PAR_NOTERM	ft_panic	The request is not yet active.
ft_err_PAR_OPEN	ft_panic	In a program, the same directory (workdir) has already been assigned to a session.
ft_err_PAR_REMOTE_ NOACCESS	ft_access	No authorization to delete in the remote system. No authorization to read attributes in the remote system.
ft_err_PAR_REMOTE_ NOTEMPTY	ft_notEmpty	The directory in the remote system is not empty.
ft_err_PAR_REMOTE_ NOTEXIST	ft_notExist	File/directory does not exist in the remote system.
ft_err_PAR_TERM	ft_ignore	The request has already terminated.
ft_err_PAR_VALUE	ft_panic	Unknown parameters/parameters are incompatible. The name of the working directory (workdir) is invalid.
ft_err_PAR_VALUE_ACCESS	ft_panic	Invalid parameter: ftamext ->accessmode.
ft_err_PAR_VALUE_AVAIL	ft_panic	Invalid parameter: ftamext ->available.
ft_err_PAR_VALUE_CANTIME	ft_panic	Invalid parameter: cantime.
ft_err_PAR_VALUE_COMPR	ft_panic	Invalid parameter: compress.
ft_err_PAR_VALUE_DIR	ft_panic	Invalid parameter: direction.
ft_err_PAR_VALUE_ENCRYPT	ft_resource	Invalid parameter: encryption.
ft_err_PAR_VALUE_FPWD	ft_panic	Invalid parameter: mgmtpasswd.
ft_err_PAR_VALUE_FTYPE	ft_panic	Invalid parameter: filetype.
ft_err_PAR_VALUE_LOCCCSN	ft_paramError	Invalid parameter: locccsn.
ft_err_PAR_VALUE_MAXREC	ft_paramError	Invalid parameter: maxrecsize.
ft_err_PAR_VALUE_PRIO	ft_panic	Invalid parameter: priority.
ft_err_PAR_VALUE_REMACC	ft_paramError	Invalid parameter: remaccount
ft_err_PAR_VALUE_REMADM	ft_auth	Invalid parameter: remadmis or invalid parameter: remadm. The user ID/transfer admission in the remote system is invalid.
ft_err_PAR_VALUE_REMCCSN	ft_paramError	Invalid parameter: remccsn.

Internal code	Error code	Description
ft_err_PAR_VALUE_REMFN	ft_paramError	Invalid parameter: remfn. The specified file does not exist/access is not permitted.
ft_err_PAR_VALUE_REMPWD	ft_paramError	Invalid parameter: rempasswd.
ft_err_PAR_VALUE_REMSYS	ft_paramError	Invalid parameter: rem. The specified remote system is unknown.
ft_err_PAR_VALUE_RFORM	ft_paramError	Invalid parameter: record format.
ft_err_PAR_VALUE_RID	ft_panic	The request ID (rid) is invalid.
ft_err_PAR_VALUE_STARTTIME	ft_panic	Invalid parameter: starttime.
ft_err_PAR_VALUE_SYNC	ft_panic	Invalid parameter: synchron.
ft_err_PAR_VALUE_TRANSP	ft_panic	Invalid parameter: transparent.
ft_err_PAR_VALUE_WMODE	ft_panic	Invalid parameter: writemode.
ft_err_PAR_VERS	ft_panic	The version of the data structure (parameter list or output area) is invalid.
ft_err_REMERR_EXIST	ft_exist	The remote file already exists.
ft_err_REMERR_INCONS	ft_remoteFile Structure	The remote file is inconsistent.
ft_err_REMERR_MEM	ft_cantCreate	The remote file is assigned no space.
ft_err_REMERR_NOACCESS	ft_access	It is not possible to access the remote file.
ft_err_REMERR_NOCREAT	ft_cantCreate	The remote file cannot be created.
ft_err_REMERR_NOTEXIST	ft_notExist	The remote file cannot be found.
ft_err_REMERR_REMADM	ft_auth	The remote transfer admission is invalid.
ft_errorEncoding	ft_resource	Code Cp850 not supported. Install extended language support for Java.
ft_errorOnCancel	ft_error	Error during abort.
ft_errorReadingCsv	ft_resource	User information could not be read.
ft_errorReadingExec	ft_resource	Default error output from script execution could not be read.
ft_errorReadingFtsadmErr	ft_resource	Default error output from fsadm command could not be read.
ft_errorWaitingForExec	ft_resource	Error waiting for the end of script execution.
ft_errorWaitingForFtsadm	ft_resource	Error waiting for the end of fsadm command.
ft_execFailed	ft_error	Script execution could not be started.
ft_exist	ft_exist	Directory already exists.
ft_exit_<exitcode>	ft_script	A script executed with executeScript has terminated with an error code.

Internal code	Error code	Description
ft_exit_255	ft_execute	Error executing executeScript.
ft_featuresNotSupported	ft_error	The parser does not support necessary features.
ft_ftsadmFailed	ft_error	IO exception on start of ftsadm.
ft_housekeepingError	ft_error	Housekeeper terminated with unknown error. Is repeated.
ft_illegalJobState	ft_error	Invalid status found on end of request.
ft_illegalMode	ft_error	Invalid mode.
ft_infoFileCantWrite	ft_resource	Info file cannot be written.
ft_InterpreterNotAlive	ft_resource	Interpreter could not be started. Execution later.
ft_invalidStartState	ft_error	Invalid status on start.
ft_jobCantCreate	ft_resource	The request could not be generated.
ft_jobCloseError	ft_resource	Error terminating request.
ft_jobExists	ft_error	This request already exists.
ft_jobFailed	ft_error	An unknown error has occurred.
ft_listDirectoryError	ft_resource	Directory cannot be read.
ft_listDirectoryFileInfo	ft_resource	Error writing persistent data.
ft_listDirectoryNoDirectory	ft_notExist	No directory:
ft_listDirectoryOutOfRangeError	ft_error	Area error on ListDirectory.
ft_listDirectoryParamError	ft_error	Parameter error on ListDirectory.
ft_listDirectorySecurity	ft_access	Access error on ListDirectory for a local directory.
ft_listDirWriterCantCreate	ft_resource	Cannot write persistent ListDirectory data.
ft_listElementUnknown	ft_error	The schema and Ftscript do not match. There is no such element in a list!
ft_listenerCantClose	ft_resource	Listener could not be terminated.
ft_lockNotReadable	ft_resource	Interpreter process not accessible.
ft_logfileCantCreate	ft_resource	Log file could not be created. No logfile, no replay.
ft_logfileCantRead	ft_resource	Log file cannot be read. Replay not possible.
ft_logfileError	ft_resource	Log file could not be created. No logfile, no replay.
ft_logfileInvalid	ft_resource	Invalid log file.
ft_logfileIoError	ft_resource	IO error for log file.

Internal code	Error code	Description
ft_logfileNoSuchFile	ft_resource	Log file could not be found.
ft_logFileNotFound	ft_resource	Log file could not be found.
ft_logfileNotReadable	ft_resource	Log file cannot be read. Replay not possible.
ft_logfileWrongVersion	ft_error	Incorrect version of the log file. This request cannot be restarted.
ft_loggerCantClose	ft_resource	Logger could not be terminated.
ft_logWrapperCantCreate	ft_resource	Log file wrapper could not be created.
ft_logWrapperCantFind	ft_resource	Log file wrapper could not be found.
ft_logWrapperCantWrite	ft_resource	Log file wrapper could not be written.
ft_mainLockError	ft_resource	Main lock cannot be set. Interpreter will be terminated.
ft_mainLockIOerror	ft_resource	Main lock cannot be set. Interpreter will be terminated.
ft_mainLockNotSet	ft_resource	Main lock not set. Interpreter will be terminated.
ft_nameGroupElementUnknown	ft_error	The schema and Ftscript do not match. There is no such name attribute.
ft_nasty_error	ft_error	An unhandled error has occurred.
ft_noInterpreter	ft_resource	Server process not present at the correct time.
ft_noKeyInLogfile	ft_error	Log file entry invalid. "key" not present. Restart no longer possible.
ft_noPartner	ft_paramError	No partner was specified for the remote file.
ft_noRef	ft_reference	Reference cannot be resolved.
ft_noRefForContextObject	ft_reference	Reference cannot be resolved.
ft_noRefInFtscript	ft_reference	Reference cannot be resolved.
ft_noScript	ft_resource	No script was specified.
ft_noSession	ft_error	openFT session could not be generated.
ft_notExist	ft_notExist	File or directory does not exist.
ft_noTransferAdmission	ft_paramError	The transfer admission is missing in the partner specification.
ft_notRepeatable	ft_error	Ftscript request cannot be repeated.
ft_noUserFaultCode	ft_paramError	Error code may not start with 'ft'.
ft_noUserScript	ft_paramError	No script was specified for executeScript.
ft_noUtf8Support	ft_resource	UTF 8 is not supported.

Internal code	Error code	Description
ft_openParamError	ft_error	Error generating the openFT session.
ft_openParamError2	ft_error	ft_open was not successful, parameter error.
ft_OrderQueueNotFound	ft_resource	Order queue could not be found.
ft_parseByteArray	ft_error	Parse or IO error.
ft_parseError	ft_error	Parse or IO error.
ft_portFileCantClose	ft_resource	PortFile could not be closed correctly.
ft_portNotFound	ft_resource	No port found. Execution later.
ft_readStatus	ft_error	Parameter error while determining status.
ft_recordSize	ft_error	Incorrect specification for maxRecSize. Defective schema?
ft_recoveryCreateDirectory	ft_recoveryFailed	Recovery failed. See manual .
ft_recoveryFailed	ft_recoveryFailed	Recovery failed. See manual .
ft_replayIllegalEntry	ft_error	Neither an error code or request ID found on replay.
ft_reqlistOutOfRange	ft_error	Error querying the openFT request list.
ft_reqstatParamError	ft_error	Incorrect parameter for reqstat. Internal error.
ft_requestIdFormat	ft_error	Error in the format of the request ID.
ft_requestIdIllegalFormat	ft_error	Error in the format for the request ID.
ft_requestInvalid	ft_error	Invalid request.
ft_resumeForbidden	ft_error	Resume not possible.
ft_resumeUnloggedTransfer	ft_resource	More than one file transfer open.
ft_rformIllegal	ft_error	Incorrect specification for RecordFormat. Defective schema?
Ft_rformOutOfRange	ft_error	Incorrect specification for RecordFormat. ftAPI version incorrect?
ft_schemaConflict	ft_error	Unknown activity, schema defective?
ft_schemaNotFound	ft_resource	Schema file *ftscript.xsd* not installed.
ft_scriptElementUnknown	ft_error	The schema and Ftscript do not match. There is no such script.
ft_scriptIdNotGiven	ft_error	Script ID not specified.
ft_scriptinfoFileCantCreate	ft_resource	Script information cannot be generated.
ft_scriptinfoFileCantWrite	ft_resource	Script information cannot be written.
ft_scriptinfoFileCantWrite2	ft_resource	Script information cannot be generated.

Internal code	Error code	Description
ft_serverSignalCommand	ft_error	Error in request acceptance by server process.
ft_sessionNotFound	ft_resource	No valid openFT session found.
ft_signalingFailed	ft_resource	Signalling to port failed. Execution later.
ft_socketClose	ft_resource	Socket cannot be closed.
ft_stateUnknown	ft_resource	Unknown status.
ft_statusCantRead	ft_resource	Status cannot be read. Request cannot be executed.
ft_statusCantWrite	ft_resource	Cannot write status.
ft_termSocketClose	ft_resource	Socket cannot be closed.
ft_timestampCantCreate	ft_resource	Timestamp could not be created.
ft_tmpFileDelete	ft_resource	Timestamp could not be deleted.
ft_tmpFileNotFound	ft_resource	Expected TmpFile was not found.
ft_traceCantCreate	ft_resource	Trace file could not be generated. No trace (deactivate trace!)
ft_traceFileCreateError	ft_resource	Trace file could not be generated. (Deactivate trace)
ft_traceFileNotFound	ft_resource	Trace file could not be found.
ft_traceSerialization	ft_error	Trace file corrupt.
ft_traceSyncFailed	ft_resource	Trace data could not be written to disk. Trace may not be complete.
ft_traceWrapperCantCreate	ft_resource	Trace file wrapper could not be generated.
ft_traceWrapperCreateError	ft_resource	Trace file wrapper could not be generated.
ft_traceWrapperNotFound	ft_resource	Trace file wrapper could not be found.
ft_transferFileIllegalSyntax	ft_error	remote/remote or local/local not permitted. Schema defective?
ft_transferMandatoryParam	ft_error	Parameter error on ft_transfer.
ft_transparentModeIllegal	ft_error	Incorrect specification in transparent mode. Defective schema?
ft_transparentModeOutOfRange	ft_error	Incorrect specification in transparent mode. ftAPI version correct?
ft_unexpectedLogEntry	ft_error	Log file invalid. Expected entry not found.
ft_unknownError	ft_error	Unknown error code.
ft_unlockOrderQueue	ft_resource	Order queue could not be released.
ft_unresolvedTmpFile	ft_reference	Local TmpFile was not found. Reference cannot be resolved.

Internal code	Error code	Description
ft_unwrap	ft_error	Unwrapping failed.
ft_userFault	ft_userFault	Error activity called in the Ftscript.
ft_userStorageCantCreate	ft_resource	Basic data cannot be initialized. EXIT, NO ERROR LOG!
ft_validatingFeaturesNot Supported	ft_error	Internal errorThe parser does not support necessary features for validation.
ft_validationError	ft_error	Validation, parse or IO error.
ft_workDirectoryPiNotFound	ft_error	PI for call directory not found in script.
ft_workDirUnusable	ft_resource	Working directory cannot be generated.
ft_writeModeIllegal	ft_error	Incorrect specification in write mode. Defective schema?
ft_writeModeOutOfRange	ft_error	Incorrect specification in write mode. Defective schema?
ft_writeOrderQueue	ft_resource	Write to order queue failed.
ft_wrongJavaVersion	ft_resource	The current Java version is too old.

The following error codes are permitted for the *faulthandler* (see section “faulthandler” on page 68).

Error codes	Description
ft_abort	Command aborted.
ft_access	Access error.
ft_admin	Administration error
ft_auth	Authentication error.
ft_cantCreate	File/directory could not be generated.
ft_cantDelete	File/directory could not be deleted.
ft_configuration	Configuration defective.
ft_connection	Connection error.
ft_corrupt	File/data/directory cannot be used.
ft_error	General error.
ft_exist	File/directory already exists.
ft_execute	Error executing executeScript.
ft_localFileStructure	Error in the local file.
ft_notEmpty	Directory is not empty.
ft_notExist	File/directory does not exist.
ft_notSupported	The functionality is not supported.
ft_panic	Severe internal error.
ft_paramError	Parameter error.
ft_paramTooLong	Parameter too long.
ft_recoveryFailed	Restart failed.
ft_reference	An Ftscript object which is not present is referenced.
ft_remoteFileStructure	Remote file structure defective.
ft_resource	Resource error (e.g. not enough storage space, no RAM)
ft_script	A script executed with executeScript has terminated with an error code.
ft_syntax	Syntax error.

Glossary

Cross-references are written in *italics*.

Activity

An *openFT script* consists of activities. These can be operating instructions to openFT (e.g. *transferFile*, *deleteFile*) or statements which control the processing flow (e.g. *parallel*, *foreach*).

Context object

Context objects are described in an *activity*'s context. Each context object possesses an ID which must be unique in the context. The context object is addressed (referenced) via this ID.

file transfer request

FT request

FT request

see *openFT request*

FTAC (File Transfer Access Controll)

Part of openFT that offers extended access protection for file transfer and file management.

FTAM-1

document type for text files

FTAM-3

document type for binary files

FTAM file attributes

All systems which permit file transfer via FTAM protocols must make their files available to their partners using a standardized description (ISO 8571). To this end, the attributes of a file are mapped from the physical filestore to a *virtual filestore* and vice versa. This process distinguishes between three groups of file attributes:

– kernel group: describes the most important file attributes.
– storage group: contains the file's storage attributes.
– security group: defines security attributes for file and system access control.

FTAM partner

Partner system which uses the *FTAM protocols* for communication.

.ftsc

File name suffix for an *openFT-Script* file.

Ftscript

Multiple logically independent openFT requests can be combined into a single request (Ftscript) using openFT-Script.

Interpreter

Here, the program which executes an *openFT-Script request*. The interpreter controls automatic *restarts*.

job

Sequence of commands, statements and data.

local system

The *FT system* at which the user is working.

openFT partners

Partner systems which communicate via openFT protocols.

openFT request

Request to an *FT system* to transfer a file from a *send system* to a *receive system* and possible start *follow-up processing requests*.

openFT script

Name of the product and the langage in which *openFT requests* are formulated.

openFT script request

The processing of an *Ftscript*.

owner of an FT request

Login name in the *local system* or *remote system* under which this *FT request* is executed. The owner is always the ID under which the request is submitted, not the ID under which it is executed.

partner system

here: *FT system* that executes FT request together with the *local system*.

password

Sequence of characters that a user must enter in order to access a user ID, file, job variable, network node or application. The user ID password serves for user *authentication*. It is used for access control. The file password is used to check access rights when users access a file (or job variable). It is used for file protection purposes.

remote system

see *Partner system*

request

here: *openFT request*

request queue

File which contains the *asynchronous requests* and their processing states. The request queue also contains the parameters set with the *ftmodo* command.

restart

Automatic continuation of an *FT request* after an interruption.

system, remote

see *partner system*

system, local

see *local system*

TCP/IP (Transmission Control Protocol/Internet Protocol)

Widespread protocol for file transfer (corresponds roughly to Layers 3 and 4 of the OSI Reference Model, i.e. Network and Transport Layer); was originally developed for the ARPANET (computer network of the US Ministry of Defense, now a de-facto standard.

Trace

Diagnostic function which logs the course of FT operation.

Transmission Control Protocol/Internet Protocol
see *TCP/IP*

Transport Name Service (TNS)
Service used to administer properties specific to transport systems. Entries for *partner systems* receive the information on the particular *transport system* employed.

Index