PyroTrans

(PyroServer, PyroBatch)

File Transfer and Directory Synchronization
over Modem, ISDN, Network and Internet

This document describes the concept behind PyroTrans. This means that this document is not a full fledged manual. Its intent is to help the user to understand the idea behind the program, to show possible uses and point out features.

If the description of a feature finds your interest, please check out the details in the help text of the software itself (especially the Script Commands help in PyroBatch).

If you intend to use PyroBatch in a project and if you have questions, please do not hesitate to contact m.schmidt@emtec.com via email.

© Copyright EmTec Innovative Software 2000 - 2002

The Concept Behind PyroTrans

PyroTrans is a software to transfer files between computers, either manually or automatically. The computers can be connected through a LAN, through the telephone system (modem, ISDN, GSM hardware) or through a IP based network (LAN or internet).

Typically such file transfers occur between a company headquarter and branch offices or field staff (where it's compression technology is especially helpful when GSM modems are used). Of course PyroTrans can also be used between a company and it's customers or to transfer data to a backup location.

The Parts of PyroTrans

PyroServer: The so called server is a purely passive component of the packet. A server waits for incoming connections, checks user rights an can send or receive files upon request. All initiative for such action however, comes from the active side of the connection, either PyroClient, PyroBatch or 3rd party Eurofile transfer software. The server can assign a directory to each user and can limit access (read, write, delete, ...) per user.

PyroServer/Pro: Same as PyroServer, but with additional functions like encryption, multiple concurrent user logins, callback option, etc.

PyroBatch: Automated file transfer. PyroBatch executes a list of predefined commands to send or receive files or execute commands on the server, etc.

PyroClient: Manually controlled file transfer. Using PyroClient a use can connect to a server and transfer or delete or rename files using the mouse or menu commands.

When using these terms however, it is important to understand, that the installation of PyroServer and PyroClient do not necessary correspond with the client and server computers of your company. Although, it is likely that the a server at the headquarter will run a PyroServer waiting for PyroBatch calls from the branch offices, but it is also possible that the headquarter runs PyroBatch to download data from PyroServers installed in the other locations.

For better understanding, PyroServer could be called PyroPassive and PyroBatch and PyroClient could be called PyroActive.

© Copyright EmTec Innovative Software 2000 - 2002

Configuration/Implementation Samples

Transfer from branch offices to headquarter (scenario 1):

Description: A program runs at night in the branch offices of a company and transfers status reports to the company headquarter.

Implementation: PyroServer is installed in the headquarter with a user entry per branch office. In the branch office at night the Windows scheduler calls PyroBatch with a predefined batch file to connect to the headquarter and send the files.

Transfer from branch office to headquarter (scenario 2):

Description: At night, a program in the headquarter calls the branch offices to download sales data.

Implementation: The branch offices run PyroServer, each with just one user entry. In the headquarter there is a PyroBatch script file for the call to the branch offices and download of the data, which are started via scheduler or via another software.

Transfer of data from field workers to headquarter:

Description: After visits to the customer, field workers transfer the orders to the sales department in the headquarter.

Implementation: The headquarter runs PyroServer. The field workers have laptops with GSM modems and a sales software. After entering the customer's order in the sales software, the software creates an order file. The employee now uses PyroClient to connect to the company and uploads the sales file via drag and drop.

Alternate Implementation: Same as above, but the sales software has a menu entry which starts PyroBatch and automatically uploads the file to the sales department..

Printing shop:

Description: Smaller customers send print jobs and designs manually while some large customers collect jobs during the day and send them automatically at night..

Implementation: The print shop runs PyroServer. The customers either use PyroClient to send a job-file for printing when necessary or collect the file in a directory and upload it it at night automatically. The server has a public user entry where all the small jobs go and a user entry (and separate directory) for each large customer.

Transfer from branch office to headquarter (scenario 3):

Description: As in scenario 1 but early in the morning the headquarter sends back files with processed status information to all branch offices.

Implementation: Headquarter and branch office have PyroClient and Server installed . During the night the branch offices make scheduled PyroBatch calls to send the information. In the morning the headquarter runs a large PyroBatch script to send back the files with processed status information.

© Copyright EmTec Innovative Software 2000 - 2002

PyroBatch Feature Overview

How to Start Scripts

PyroBatch scripts can be started manually from PyroBatch's menu or directly from the command line as `PYROBATCH /EXEC:<scriptname>`

Overview of script-commands

Connect:	connect to PyroServer
Disconnect:	end connection
RequestCallback	request a call back from the server
LocalChDir:	change directory (on the PyroBatch file system)
RemoteChDir:	change directory (on the PyroServer file system)
LocalRename:	rename file on the PyroBatch file system
RemoteRename:	rename file on the PyroServer file system
LocalDelete:	delete file on the PyroBatch file system
RemoteDelete:	delete file on the PyroServer File system
Get:	get file from PyroServer
Put:	send file to PyroServer
GetMove:	get file from PyroServer and delete remote file
PutMove:	send file to PyroServer and delete locally
GetDir:	copy whole directory from PyroServer
PutDir:	send whole directory to PyroServer
GetSync:	synchronize directory between remote and local
PutSync:	synchronize directory between local and remote
LocalExec:	execute local command (on PyroBatch machine)
RemoteExec:	execute remote command (on PyroServer machine)
OnError:	select error handling
ForEach:	executed command multiple times
Milestone:	make entry in milestone log
SetRetry:	set retry count and delay for failed commands
TerminateAfterScript:	select program termination after end of script
Comments:	comments are placed behind //, # or ;

A detailed description to each command is available from the help menu of PyroBatch.

© Copyright EmTec Innovative Software 2000 - 2002

Script Style

The PyroBatch script language uses a flexible syntax which allows you to write commands in various formats. This lets you write scripts in a form which is similar to other program languages you may know, e.g. Windows Batch Files, Visual Basic, Perl.

The following samples are all legal forms of a PutMove command:

```
PUTMOVE SALES.DAT SALES2.DAT
PutMove "Sales.dat", "Sales2.dat"
PutMove("Sales.dat", "Sales2.dat")
```

Definition and Use of Command Line Macros

If a similar script is necessary for multiple calls, it is possible to use a common script and insert macros like `$(phonenumber)` or `$(password)` in the places in which the hosts require different values. When starting PyroBatch it is possible to define the actual values for these as in

```
PYROBATCH /EXEC:transfer.cmd /D:phonenumber=5554467855 /D:password=secret
/D:hostname=branch01
```

The transfer script might then look like this:

```
TerminateAfterScript 1
Connect $(phonenumber) pyrotrans $(password)
LocalChDir c:\incoming\$(hostname)
RemoteChDir outgoing
GetMove sales.dat $(hostname).dat
Disconnect
```

Error Handling

If a command produces an error, the script will immediately be aborte and the current connection (if any) terminated. The reason for the error will be documented in the script log. Error codes and error classes (groups of error codes) are described in the PyroBatch online help.

It is possible to tolerate errors in specific commands by putting a dash in front of the command name (e.g. `-LocalDelete "alt.dat"` .if a file is to be deleted that may or may not exist.) Further, the error handling can be controlled by the `OnError` command (see the examples below).

Retrying Failed Commands

If an error occurs in a script (and the script terminates with an error code) that was started from the command line, you can add the `/RETRY:` parameter on the command line to specify that the whole script should be executed again.

Alternately, within a script it is possible to precede a command with an @ character, to retry

© Copyright EmTec Innovative Software 2000 - 2002

just this command if it fails (e.g. if making a connection fails because it is busy). The number of retries and their delay can be configured inside the script with the `SetRetry` command.. However, the use of @ only makes sense, if you expect that a failed command may succeed later.

Logging

Script log: For each run of a script PyroBatch will write a detailed log (multiple entries per executed command including result and possible sub-results). The log is formatted in a standardized way which can easily be parsed by another application (e.g. by a sales application to determine if the transfers went ok or not). The format itself is described in the online help.

Milestones: If necessary (or desired) the script can generate a so called milestone file which logs the state of the script at certain important points of the script (e.g. a script that calls multiple branch offices could mark the success at the end of processing each office). Milestone files have only a few entries, in fact exactly one line per milestone command and can very easily be scanned or parsed to determine the success of a script.

Script-Processing with PyroBatch

Sample Scripts for Configuration Samples

Headquarter with Branch Office (Scenario 1):

This script calls the headquarter and uploads the local file called sales.dat as location1.dat and then deletes it on the local computer.

```
// Call Headquarter, and Login as location1
Connect 05553456789 location1 secret

// Upload File sales.dat with name location1.dat
LocalChDir c:\outbox
-LocalDelete location1.dat
LocalRename sales.dat location1.dat
Put location1.dat
LocalDelete location1.dat

// Endconnection
Disonnect
```

Headquarter with Branch Office (Scenario 1), alternate:

Same but more elegant.

```
Connect 05553456789 location1 secret
PutMove c:\outbox\sales.dat location1.dat
Disonnect
```

Headquarter with Branch Office (Scenario 2):

Headquarter calls three locations and downloads their sales.dat as location1/2/3.dat.

```
// Call from Headquarter to location 1
// Download of the file sales.dat with name location1.dat
Connect 05553456789 comanyabc secret
GetMove sales.dat c:\inbox\location1.dat
Disconnect


// Call from Headquarter to location 2
// Download of the  file sales.dat with name location2.dat
Connect 05556666666 comanyabc secret
GetMove sales.dat c:\inbox\location2.dat
Disconnect


// Call from Headquarter to location 3
// Download of the file sales.dat with name location3.dat
Connect 05559876543 comanyabc secret
GetMove sales.dat c:\inbox\location2.dat
Disconnect
```

Headquarter with Branch Office (Scenario 2) Ignoring Error (Excerpt):

© Copyright EmTec Innovative Software 2000 - 2002

This variant of the example above uses `OnError` to make sure that an error in one location will not abort the processing of the whole script (and thus ignore the later locations).

```
OnError Ignore

Connect 05553456789 comanyabc secret
GetMove sales.dat c:\inbox\location1.dat
Disconnect

    (rest of script as above)
```

Headquarter with Branch Office (Scenario 2) with Error Handling (Excerpt):

This example uses `OnError SkipTo` to skip to the next location when an error occurs.. `OnError` is used in a way that `Disconnect` will be processed or skipped, depending on the online state (i.e. depending on if connect was successful).

```
// Call of Headquarter to location 1
OnError SkipTo NaechsteLocation
Connect 05553456789 comanyabc secret

OnError SkipTo Endconnection
GetMove sales.dat c:\inbox\location1.dat
; more things to do online could be done here
:Endconnection
Disconnect



:NaechsteLocation
// Call of Headquarter to location 2
OnError SkipTo NaechsteLocation
Connect 0555666666 comanyabc secret

OnError SkipTo Endconnection
GetMove sales.dat c:\inbox\location2.dat
; more things to do online could be done here
:Endconnection
Disconnect



:NaechsteLocation
```

```
// Call of Headquarter to location 3
OnError SkipTo Ende
Connect 05559876543 comanyabc secret

OnError SkipTo Endconnection
GetMove sales.dat c:\inbox\location2.dat
; more things to do online could be done here
:Endconnection
Disconnect

:Ende
```

Headquarter with Branch Office (Scenario 2) with Elegant Error Handling:

This example solves the same problem as above, but does away with most of the `OnError` commands by using a trick. It also uses `OnError SkipTo` to skip to the next location.

If an error occurs it jumps to the next ":Nextlocation" label. ":Nextlocation" has been placed before the `Disconnect` command, to terminate the connection if the error occurs while we are online. However, if the error occurs in the connect command the program would not be in online state when jumping to the disconnect and disconnect would produce an error on its own (causing the program to yet again jump down to the next label). However, the error to `disconnect` while the program is not online, is tolerable (in fact in this solution it is somewhat planned) so a hyphen is placed before the disconnect command to ignore it's error state (in other words, disconnect if online but do not complain if you are already offline).

```
OnError SkipTo Nextlocation


// Call of Headquarter to location 1
// Download der File sales.dat with name location1.dat
Connect 05553456789 comanyabc secret
GetMove sales.dat c:\inbox\location1.dat
:Nextlocation
-Disconnect


// Call of Headquarter to location 2
// Download der File sales.dat with name location2.dat
Connect 05556666666 comanyabc secret
GetMove sales.dat c:\inbox\location2.dat
:Nextlocation
-Disconnect
```

© Copyright EmTec Innovative Software 2000 - 2002

```
   // Call of Headquarter to location 3
   // Download der File sales.dat with name location3.dat
   Connect 05559876543 comanyabc secret
   GetMove sales.dat c:\inbox\location2.dat
   :Nextlocation
   -Disconnect
```

Headquarter with Branch Office (Scenario 2) with Error Checking and Milestones:

The error handling is the same as in the previous example (see above).
The `milestone` commands before each `:Nextlocation` label will write the error status to file. This is either `#200 OK` if all went well (error status from `GetMove`) or the error (eg. `#400 BUSY`) which caused `OnError` to skip to the label (which passes and processes the `Milesone` command on the way down).

```
   OnError SkipTo Nextlocation
   Milestone Filename transfer.log


   // Call of Headquarter to location 1
   // Download der File sales.dat with name location1.dat
   Connect 05553456789 comanyabc secret
   GetMove sales.dat c:\inbox\location1.dat
   :Nextlocation
   -Disconnect
   Milestone Location1


   // Call of Headquarter to location 2
   // Download der File sales.dat with name location2.dat
   Connect 05556666666 comanyabc secret
   GetMove sales.dat c:\inbox\location2.dat
   :Nextlocation
   -Disconnect
   Milestone Location2


   // Call of Headquarter to location 3
   // Download der File sales.dat with name location3.dat
   Connect 05559876543 comanyabc secret
   GetMove sales.dat c:\inbox\location2.dat
   :Nextlocation
   Milestone Location3
```

© Copyright EmTec Innovative Software 2000 - 2002

© Copyright EmTec Innovative Software 2000 - 2002