Cyber-Ark® Software

Distribution and Collection Agent Implementation Guide

The Cyber-Ark® Vault

Version 4.1

DCA004-1-0-2

Table of Contents

Introduction

The Cyber-Ark Distribution and Collection Agent is a file-transfer management system that enables reliable and secure data transfer between organizations and their partners, suppliers and customers, providing a crucial platform for business operation and continuity.

The Distribution and Collection Agent enables you to control all aspects of movement of your business data between any two entities, and provides you with a combination of the following unique features:

- **Secure file transfer** based on Cyber-Ark's patented Vaulting technology
- **Flexibility** to support file transfer processes in which any business logic can be deployed
- **Easy integration** with Legacy or Enterprise systems
- **High performance and scalability**
- **Reliability** that ensures delivery
- **Easy process management** enables you to add new process and trading partners
- **Auditing and monitoring** that generates log files and enables you to track transferred files' activity

Secure File Transfer

The Distribution and Collection Agent facilitates secure file transfer between entities. Each file transfer process benefits from the standard security layers of the Vault, which include the following:

- Data encryption for files stored in the Vault as well as those being transferred.
- Variety of authentication mechanisms, including username/password, PKI, SecurID, etc.
- Auditing
- Additional Vault features

Ensured Delivery

The Distribution and Collection Agent ensures that files reach their destination, while constantly maintaining data integrity. Regardless of network interferences, the Distribution and Collection Agent automatically attempts to provide checkpoint/restart processes by reconnecting and resubmitting the files until they reach their destination.

The system can also resume large file transfers that were interrupted in the middle, rather than starting them from the beginning, to complete transfers quickly.

This is a step above FTP and HTTP which sometimes compromises data integrity due to network interferences and protocol inability to recover from these interferences.

Authentication

The Distribution and Collection Agent logs onto the Cyber-Ark Vault with a user who is already specified in the Vault. The Vault supports the following authentication methods for the Distribution and Collection Agent:

- **Password** – A username and password of a user who is already specified in the Vault.
- **Token** – A password that has been encrypted by a key on a USB token or a Smartcard.
- **PKI** – A PKI certificate.
- **Radius** – Logon through RADIUS authentication using logon credentials that are stored in the RADIUS server.

The username and encrypted password are stored in a user credentials file which the Distribution and Collection Agent accesses to log onto the Vault and carry out file transfers.

Access control

Users who are given access to a Safe are called Safe Owners. A user who is not a Safe owner of a particular Safe, will not be able to access to it.

Permissions are given to Safe owners to determine the tasks that the user can carry out in the Safe. Safe owners can be given different permissions in each Safe that would enable them to carry out different tasks in each one. The user that the Distribution and Collection Agent uses to log onto the Vault must be a Safe Owner of the Safes to which or from which files will be transferred. In addition, they must have the appropriate permissions to carry out the transfer.

This combination of Safe ownership and permissions within Safes ensures that specific accounts will be used for file transfers, and enables the Vault to track every activity that the user carries out.

Monitoring and File-level tracking

The Distribution and Collection Agent creates detailed log files that describe the different aspects of the file transfer processes.

In addition, all files that are stored in the Cyber-Ark Vault benefit from the Vault's activity monitoring and tracking features.

Overview

The Distribution and Collection Agent performs file transfer operations based on pre-defined file transfer processes. These processes are configured in XML files which describe the file transfer process logic.

The process logic defines the process attributes and specifies the customers, partners or suppliers with whom the file transfer will exchange data.

Each file transfer process is configured in a single XML file. These processes can later on be monitored on a process level, customer/partner level and file level.

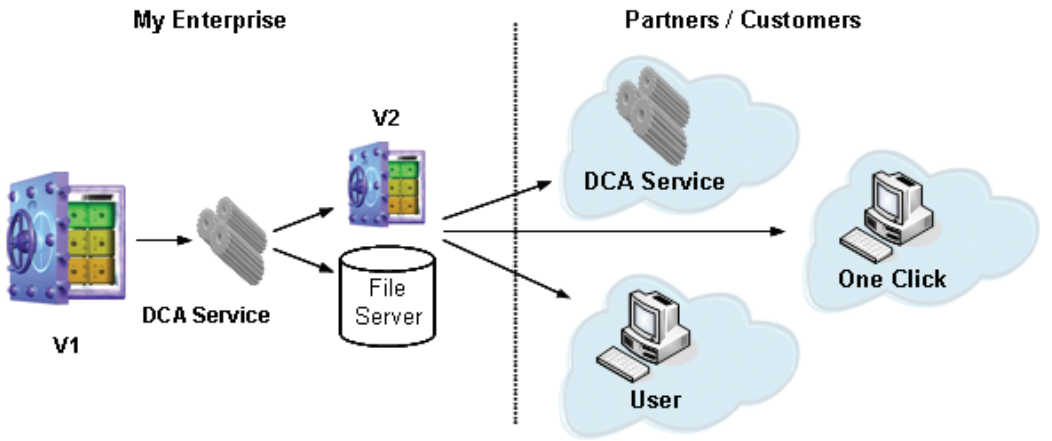
Process configuration defines the attributes of the transfer process, including the following major functionalities:

- **Source and destination path definition** – Source and destination paths may specify either Vault paths or file system paths.
- **File selection** – Files that will be transferred by a process can be selected either by wildcards and/or by file categories, if they are transferred from a Vault. In addition, file selection can be either specific or recursive.
- **Source files control** – Also known as the non-duplication mechanism, this feature determines what will happen to the source files after they have been transferred successfully. This can ensure that the same file will not be transferred twice.
- **Delivery options** – File transfers that are not completed successfully can be repeated at pre-determined intervals and for a set number of times. In addition, these options determine the action to carry out if the file already exists in the destination.
- **Scheduling** – Processes can be run automatically at pre-determined intervals or when activated by a Command Line Interface. In this way, they can be included in an external scheduler and run as part of a larger business process.

Transferring Files Securely

File Transfers from a Vault

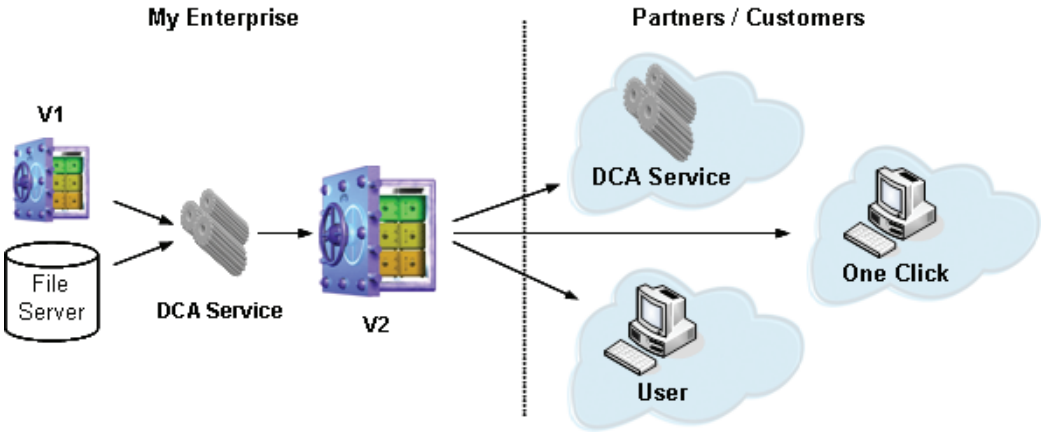
The following diagram explains the flow of information that takes place during a file transfer from a Vault through the Distribution and Collection Agent to either a Vault or a file server.



The DCA takes the source files from the Vault (V1) and transfers it to a destination location in either a Vault (V2) or a file system. From there, the files can be transferred onwards to another destination.

File Transfers to a Vault

The following diagram explains the flow of information that takes place during a file transfer from either a Vault or a file server through the Distribution and Collection Agent to a Vault.



The DCA takes the source files from either a file system or a Vault (V1) and transfers it to a destination in a Vault (V2). From there, the files can be transferred onwards to another destination.

Installing the DCA

Requirements

The minimum system requirements for the Cyber-Ark Distribution and Collection Agent are as follows:

Platform:	Pentium IV 2.0Ghz (with CD-ROM)
Disk space:	50MB of free disk space on the local file system
Memory:	512MB

The Cyber-Ark Distribution and Collection Agent is currently supported on the following platforms:

- Windows XP
- Windows 2000
- Windows 2003

In order to support Perl user exits, the following version of Perl must be installed:

- Perl 5.8.*

Cyber-Ark Vault

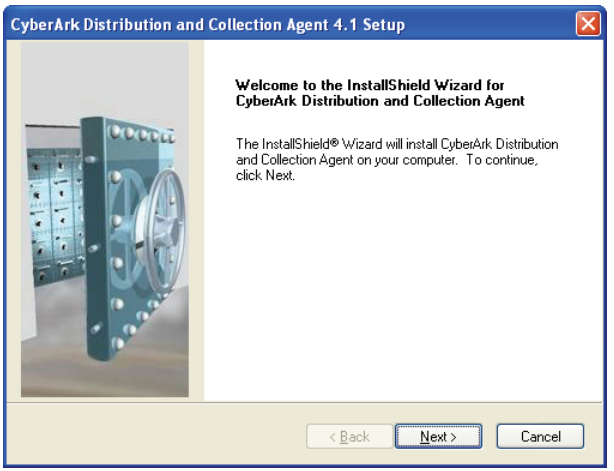
- The Cyber-Ark Distribution and Collection Agent works with the Cyber-Ark Vault, version 3.50 or higher.

Installation

The Cyber-Ark Distribution and Collection Agent component is on an installation CD that you will receive from your Cyber-Ark representative. It can be installed directly from the CD.

Note: The Windows service for the Distribution and Collection Agent component is **CyberArk Distribution and Collection Agent**.

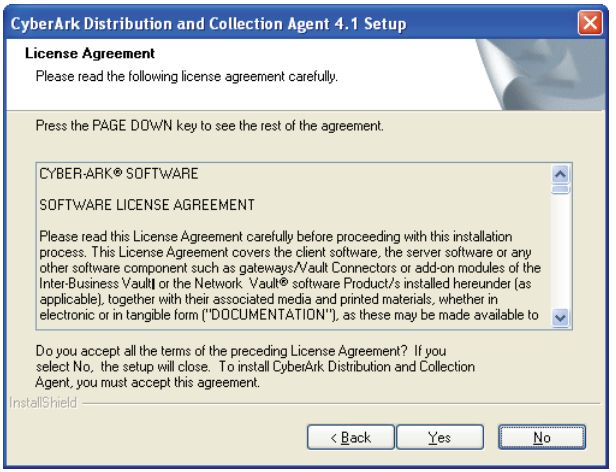
1. From the Distribution and Collection Agent installation CD, double-click **Setup.exe**; the installation process begins and the Setup window appears.



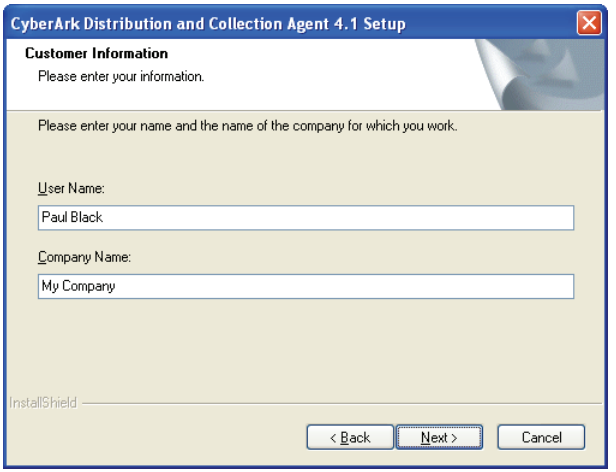
2. If you have not already closed any open Windows applications, it is strongly advised that you do so at this point.

   Note: You can exit installation at any time by clicking **Cancel**. You can return to the previous installation window by clicking **Back**, where applicable.

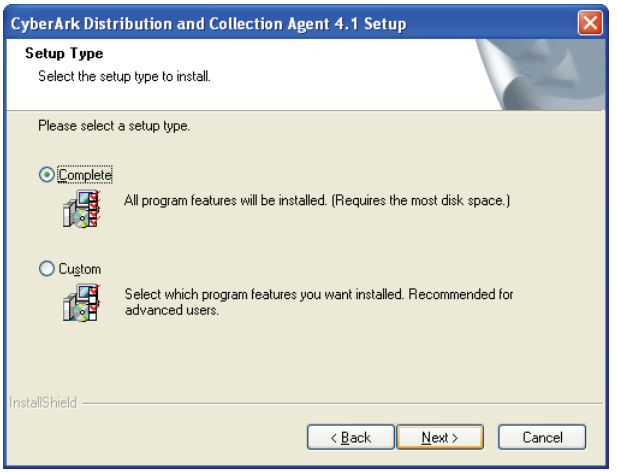
3. Click **Next** to proceed to the next step of the installation, which enables you to view the Cyber-Ark license and accept the terms of the License Agreement.



4. Click **Yes** to accept the Cyber-Ark License Agreement and proceed to the User Information window, which enables you to enter user information.



5. Enter your name and Company name in the appropriate fields, then click **Next** to proceed to the Setup Type window which enables you to choose whether to install the complete Distribution and Collection Agent or specific components.

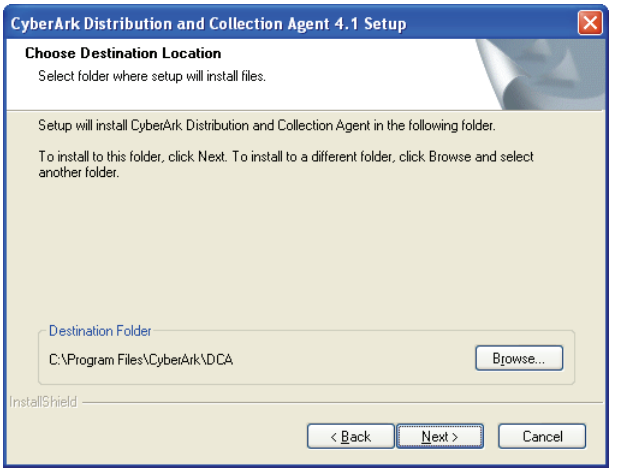


6. Select **Complete** to install all program features, then click **Next**, and proceed to step 10,

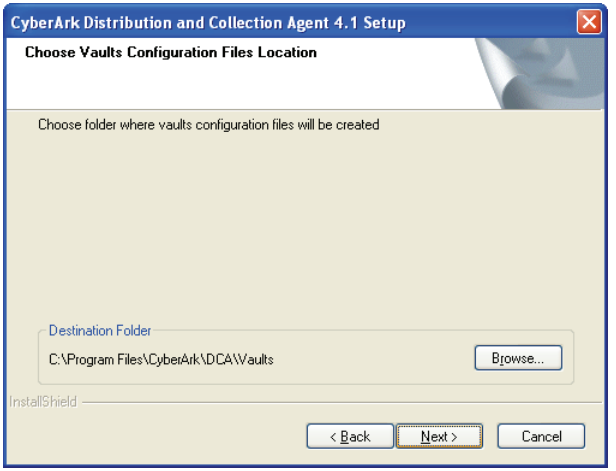
   or,

   Select **Custom** to specify which features to install, then click **Next** to display the Destination Location window, as shown below.

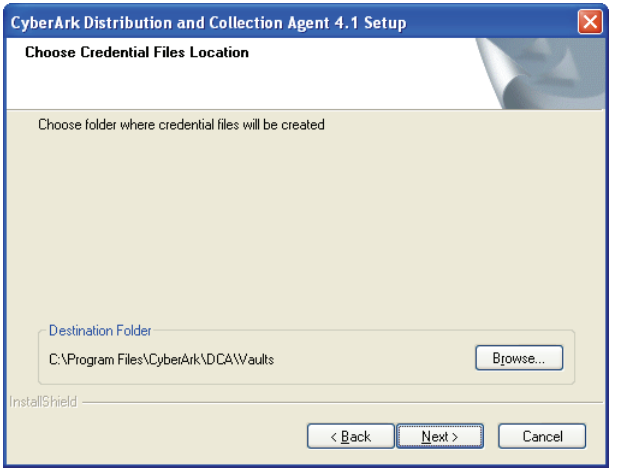
   Note: The complete installation includes the Perl user exits feature. In order for this feature to be installed successfully, Perl must be installed on your machine before installation.



7. Click **Next** to accept the default location provided by the installation, or,
   Click **Change** to select another location, then click **Next**.
   The Vaults Configuration Files Location window appears.

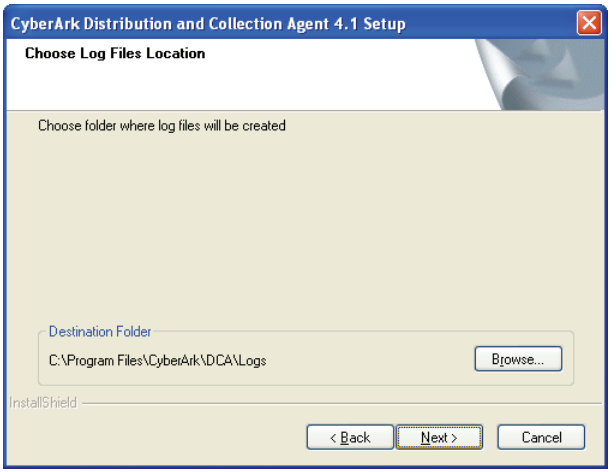


8. Click **Next** to accept the default location provided by the installation,
   or,
   Click **Change** to select another location, then click **Next**.
   The Credential Files Location window appears.

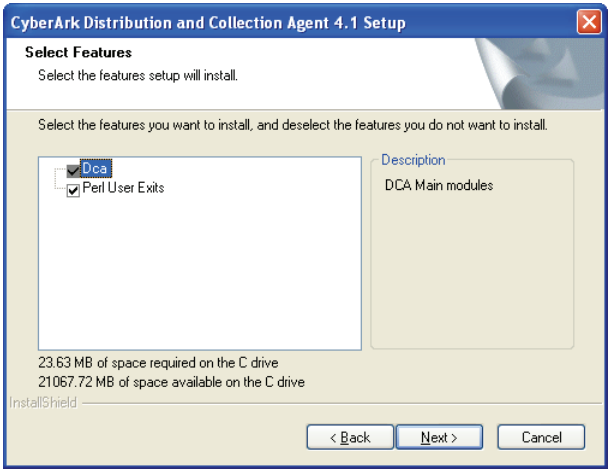


9. Click **Next** to accept the default location provided by the installation,
   or,
   Click **Change** to select another location, then click **Next**.

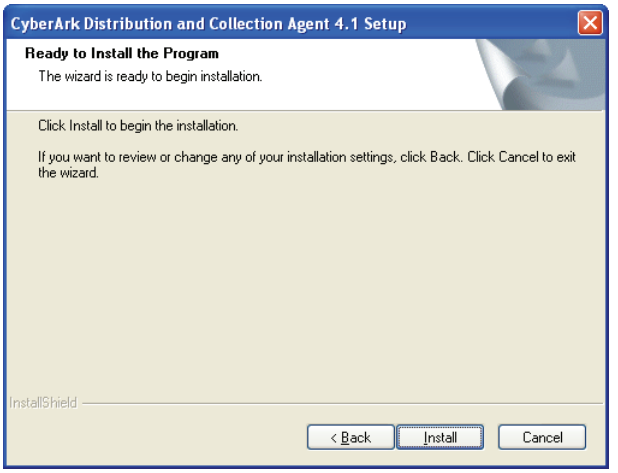
The Log Files Location window appears.



10. Click **Next** to accept the default location provided by the installation,
    or,
    Click **Change** to select another location, then click **Next**.
    The Select Features window appears.

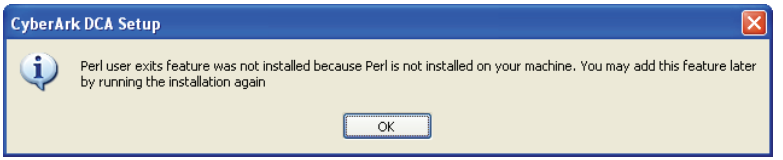


11. By default, both the features in the list are selected. Deselect the feature(s) you do not wish to install, then click **Next**; the Ready to Install window appears.

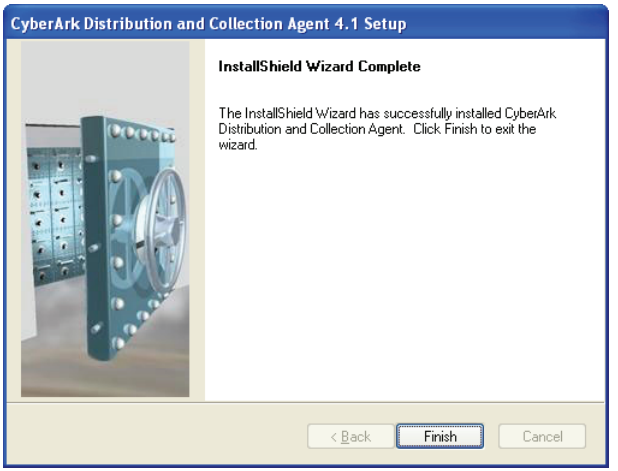


12. Click **Install** to start installation.

    If you chose to install a complete installation in the Setup Type window, or you chose to install Perl User Exits in the Select Features window, but Perl is not installed on your machine, the following message will appear.



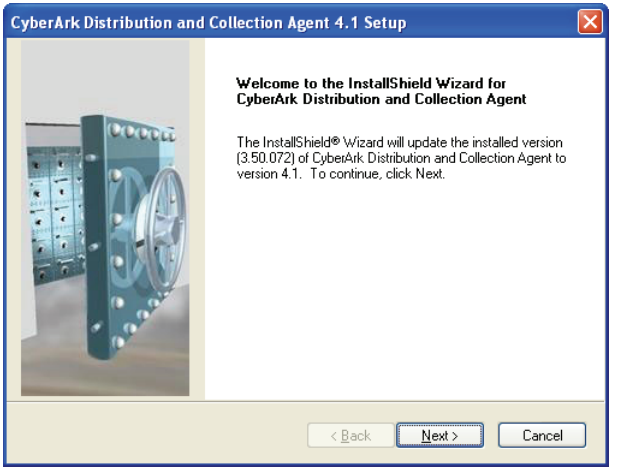
13. Click **OK**; the Setup Complete window appears.



14. Click **Finish** to complete the Distribution and Collection Agent installation.

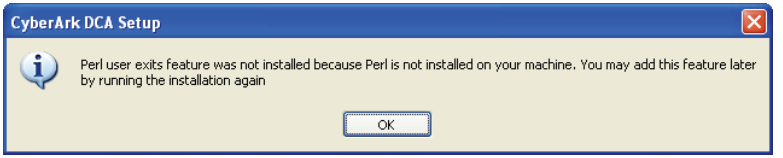
Upgrading

1. From the Distribution and Collection Agent installation CD, double-click **Setup.exe**; the installation process begins and the Setup window appears.

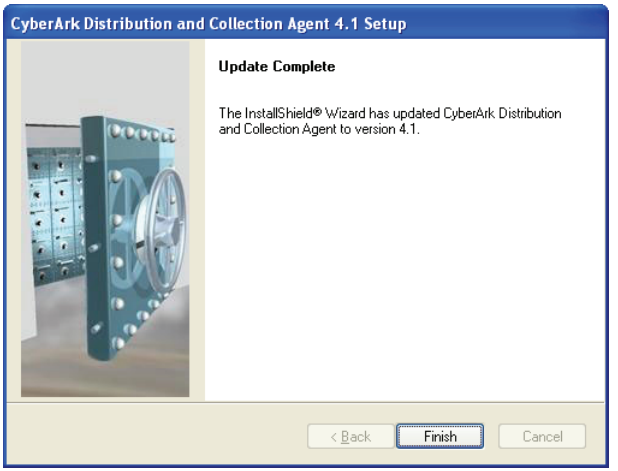


2. Click **Next** to proceed to the next step of the installation, which upgrades the DCA configuration files.

   If Perl is not installed on your machine, the following message will appear.



3. Click **OK**; the Setup Complete window appears.



4. Click **Finish** to complete the Distribution and Collection Agent installation.

DCA Components

The following components are installed during the Distribution and Collection Agent installation:

Executables

- **DCA.exe** – The DCA executable is the main module of the Distribution and Collection Agent. It enables the user to run file transfer processes and use all the features described in this document.
- **CreateCredFile.exe** – The CreateCredFile executable enables you to create a credentials file that contains a user's logon credentials. The Distribution and Collection Agent can use this file to logon to the Vault automatically during a transfer. For more information, refer to *Appendix D: Creating a User Credential File*, page *120*.
- **Dcacli.exe** – The DCACLI executable enables the user to run processes from a command line interface. For more information, refer to *Running Processes from a Command Line Interface*, page *100*.

Configuration Files

- **Dca parameter file** – The Dca parameter file specifies the parameters that determine how the Distribution and Collection Agent will work. For more information, refer to *Appendix A: Dca.ini*, page *104*.
- **Vault parameter file** – The Vault.ini file contains all the information about the Vault that the Distribution and Collection Agent will access during a transfer. For more information, refer to *Appendix B: Vault.ini,* page *107*.
- **Credentials file** – The user credentials file contains the logon credentials of a Vault user that will be used by the Distribution and Collection Agent to access the Vault during a transfer. For more information, refer to *Appendix D: Creating a User Credential File*, page *120*.

Folders

- **Temp** – The Temp folder is used during file transfers as a temporary storage location.
- **Logs** – The Logs folder contains all the log files created by the Distribution and Collection Agent.
- **Processes** – The Processes folder contains the configuration files of all the processes that will be executed by the Distribution and Collection Agent.

Administrating the Distribution and Collection Agent

The Distribution and Collection Agent is installed on a Windows system as an automatic system service called **CyberArk DCA**.

It can be stopped and started through the standard Windows service management tools. After installation, the service is started automatically.

To Stop the CyberArk DCA Service

1. From the **Start** menu, select **Settings**, then **Control Panel**.
2. From the list of Control Panel options, select **Administrative Tools**, then **Services**; the Services window appears.
3. Right-click **CyberArk DCA**, and select **Stop**.

To Start the CyberArk DCA Service

1. From the **Start** menu, select **Settings**, then **Control Panel**.
2. From the list of Control Panel options, select **Administrative Tools**, then **Services**; the Services window appears.
3. Right-click **CyberArk DCA**, and select **Start**.

Planning File Transfer Processes

Planning is the most important stage of implementing the Cyber-Ark Distribution and Collection Agent in your organization. This chapter discusses various issues that must be considered before installation, and affects how you install and configure the Distribution and Collection Agent.

During the planning stage, you gather information about your file transfer processes and define exact requirements regarding transfers and the users who define and initiate them.

The architecture and flow of your file transfer processes determine how and where the Cyber-Ark Vault will be installed, and how the Vault features will be configured.

Although time-consuming, successful planning will ensure that the Distribution and Collection Agent will be implemented effectively and successfully in your organization.

Stage 1 – Architecture

Prepare an architectural design of your file transfer process, starting with the existing application that creates the data that will be distributed and ending with the application that will collect it.

Use Cyber-Ark Vaults and the Distribution and Collection Agent to create links between different networks. Plan the locations of the Vaults according to your network infrastructure, and the Distribution and Collection Agents as the links between them that build the file transfer chain.

Stage 2 – Preparing Users and Safes

In the Vault, create the users for end users and for the Distribution and Collection Agents that will carry out the file transfer processes, and specify their authentication method. Create Safes to store the files that will be transferred, and specify the relevant Safe properties. Give each user ownership and access permission on the Safes he will need to access to carry out file transfers.

Prepare a user credentials file for each user who will carry out a transfer to or from the Vault. This file will contain the user's Vault username and logon credentials so that the user can access the Vault automatically and carry out the process.

Stage 3 – Define the File Transfer Process

Define the different steps of the file transfer process. For example, if a file transfer will move files from a local file system in one network to a Vault in another network, the following steps would describe the process:

1. Upload the files to transfer from the local file system on Network A to the internal Vault.
2. Move the files from the internal Vault to the external Vault.
3. The transferred files will be collected manually by an end user or by a third party application or another DCA.

Stage 4 – Plan the Process for each Step

Define the following details of the process:

1. Is the source path of the files a file system or a Vault?
   If it is a Vault:
   i. Which Safe will these files be taken from?
   ii. Which user will retrieve the files from the Vault?
2. Is the destination path of the files a file system or a Vault?
   If it is a Vault:
   i. Which Safe will these files be delivered to?
   ii. Which users will store the files in the Vault?
3. What should be done with source files after they have been transferred?
4. What action should be taken if a destination file already exists?
5. How will the process be activated?
   i. Will the Distribution and Collection Agent activate the process at a specified interval?
   ii. Will the process be activated by a command-line interface?

Stage 5 – Considerations for Integrating with Existing Applications

1. Do you require email notifications or other types of notifications?
2. Do the applications that create the data need to transfer metadata with the file?
3. Do you need to set the file's destination dynamically according to its content, name, or other parameters?
4. Do you need special operations to be executed before or after a file has been transferred?
5. Do you need special operations to be executed before or after a process has completed?
6. Do you need special operations to be executed before or after a group of files has been transferred to a customer/partner?

Configuring File Transfer Processes

A process is a series of parameters that define a transfer that is carried out by the Distribution and Collection Agent. These parameters determine the type of transfer, the source and destination of the transfer, and any additional behavior that will be carried out during the transfer.

Process Definition Files

The process parameters are stored in the DCA\Processes folder in process definition files called <process>.xml. Within these process definition files, the parameters are grouped in three sections, as follows:

- **General Process Attributes** – These parameters set the general process behavior regarding scheduling, file selection, etc. For a list of required parameters, refer to *General Process Attributes*, page *30*. For a full list of parameters, refer to *Appendix C: Process Definition Parameters*, page *109*.
- **Rules** – The rule indicates the source and destination paths of the file(s) to transfer. One or more rules can be defined for each process.
- **User exits** – These optional parameters specify a user-defined program that customizes the file transfer process at various stages of the process.

The structure of a process definition file looks like this:

```
<Process>
       .
       General process attributes
       .
       <Rules>
              Transfer rules section
       </Rules>
       <UserExits>
              User exits section
       </UserExits>
</Process>
```

General Process Attributes

The general process attributes specify the process' general parameters, such as the name and type of the process. For a full list and explanation of these parameters, refer to *Appendix C: Process Definition Parameters*, page *109*.

The following table explains the parameters that must be defined in order for a process to run:

Parameter	Indicates …
ProcessName	The name of the process.
ProcessEnabled	Whether or not the process is active and will run, either automatically at regular intervals according to a predetermined period of time, or when activated by an external catalyst. This is defined in the ProcessScheduling parameter.
ProcessType	The type of process that will be carried out by these parameters. Possible options are: ◆ **N2N** – A transfer in which the rules with their source and destination are specified. ◆ **Automap** – A transfer that maps the destination automatically based on minimum details. The type of files transfer are specified in the AutomapPattern parameter.
NonDuplicationMethod	How the source file(s) will be handled after the transfer has been completed.
NonDuplicationMethod OnFailure	How file(s) will be handled after the transfer, if the transfer failed.
OnFileExistsInDest	The action that will take place when files are transferred to a destination location that already contains files of the same name.
OnFileExistsInArchive	The action to take if a file is moved to the archive folder after being transferred, but a file with the same name already exists there.
OnFileExistsInFailure Archive	The action to take if files are moved to the failure archive folder after the transfer failed, but a file with the same name already exists in that folder.

Parameter	Indicates …
ProcessScheduling	Whether the process will start automatically at regular intervals according to a predetermined period of time, or whether an external catalyst will activate the transfer.
ProcessRecursive	Whether or not the contents of subfolders of the specified folders will be included in the transfer.
FileSelectionPattern	A file pattern that indicates which files to transfer during a process.
FileObjectsEarlyDelete	Used during massive file transfers to allow internal file objects that are allocated in memory to be purged early in the process in order to reduce memory consumption.

Specifying Static or Dynamic Rules

During a process, files can either be transferred between specified paths in a file server or a Vault, or locations that are specified dynamically by the process.

This activity is specified by the **ProcessType** parameter. The following table lists the available options:

Parameter	Indicates …
N2N	The process will transfer files from a specified source location to a specified destination location. Each transfer from a specified source to a specified destination is defined in a rule. An N2N process can have one or more rules.
Automap	The process will create transfer rules dynamically, based on the source folders structure and other minimum definitions.

For more information about creating transfer rules dynamically, see *Creating Rules Dynamically*, page *54*.

Controlling Source Files

After a file has been transferred to a destination location, the original file can be left as it is, deleted, or moved to a new location, regardless of whether if it is in a Vault or in a local file system. If it is in a Vault, it can be marked with an Access Mark. If the file is in a file system, its archive attribute can be set in its file properties. For more information about Access Marks, refer to the Cyber-Ark User's Guide.

This activity is specified by the **NonDuplicationMethod** parameter. The following table lists the available options:

Parameter	Indicates …
Delete	The source file will be deleted from the source location.
Move	The source file will be moved to the folder specified in the ArchivePath parameter in the SourcePort parameters.
AccessMark	If the source file is in a Vault, the file will be marked with a 'retrieved' access mark. This ensures that only files that are marked with a 'new' or 'modified' access mark will be transferred.
ArchiveBit	If the file is in a file system, its archive attribute can be set in its file properties. This ensures that only files whose archive attribute is not set will be transferred.
None	No action will be carried out on the source file after it has been copied to a destination location. **Note:** The next time the transfer will run, the file will be copied again because the Non-duplication parameter is set to 'None'.

The action that will be taken if the transfer process fails is specified in the **NonDuplicationMethodOnFailure** parameter. The following table lists the available options:

Parameter	Indicates …
Delete	The source file will be deleted from the source location.
Move	The source file will be moved to the folder specified in the ArchivePath parameter in the SourcePort parameters.
AccessMark	If the source file is in a Vault, the file will be marked with a 'retrieved' access mark.
ArchiveBit	If the file is in a file system, its archive attribute can be set in its file properties.
None	No action will be carried out on the source file after it has been copied to a destination location. **Note:** The next time the transfer will run, the file will be copied again because the Non-duplication-on-failure parameter is set to 'None'.

Controlling Transferred Files

You can also specify the action that will take place when files are transferred to a destination location that already contains files of the same name. This enables you to determine whether existing files will be overwritten or not, and whether a message will be written to the process log file.

This activity is specified by the **OnFileExistsInDest** parameter. The following table lists the available options:

Parameter	Indicates …
KeepExisting	The file that is being transferred will not be written to the destination location, so that it doesn't overwrite the file that is currently there. No error will be issued to the log file.
KeepExisting WithWarning	The file that is being transferred will not be written to the destination location, so that it doesn't overwrite the file that is currently there. In addition, a warning will be written in the process log.
Overwrite	The new file is copied to the destination location, overwriting the existing file. No error will be issued to the log file. **Note:** In previous versions, when an overwrite action was performed an informatory message was logged. However, from this version, overwrite actions are no longer logged as informatory. To log an overwrite action as a warning, specify 'OverwriteWithWarning'.
OverwriteWithWarning	The new file is copied to the destination location, overwriting the existing file, and a warning will be written in the process log.
Fail	The file transfer will fail and an error will be issued to the process log.
Retry	The Distribution and Collection Agent will keep trying to transfer the file to the destination location without overwriting the existing file. After repeating the transfer the number of times specified in the **FilesRetries** parameter, a failure log message will be written to the process log.

Parameter	Indicates …
Skip	The file that is being transferred will not be copied to the destination location, so that it doesn't overwrite the file that is currently there, and the non-duplication method will not be executed on the source file. No error will be written in the process log.
SkipWithWarning	The file that is being transferred will not be copied to the destination location, and the non-duplication method will not be executed on the source file. In addition, a warning will be written in the process log.
UserExit	The user exit **OnFileExistsInDestination** will be executed. After the user exit is finished, the file that is being transferred will be copied to the destination location, overwriting the existing file if it is still there.

After files have been transferred, the source files can be moved to the archive folder. If files with the same name already exist in the archive folder, one of several actions can be taken. These actions are specified by the **OnFileExistsInArchive** parameter. The following table lists the available options:

Parameter	Indicates …
Overwrite	The new file will be stored in the archive folder and will overwrite the file that is already stored there.
OverwriteWith Warning	The new file will be stored in the archive folder and will overwrite the file that is already stored there. In addition, a warning will be written in the process file.
Fail	The new file will not be stored in the archive folder and an error will be written to the log file.
UserExit	The new file will not be stored in the archive folder and a user exit called **OnFileExistsInArchiveFolder** will be called automatically.
UserExitWith Overwrite	A user exit called **OnFileExistsInArchiveFolder** will be called automatically and then the file in the archive folder will be overwritten.

When a transfer process fails, source files can be moved to the failure archive folder. If files with the same name already exist in the failure archive folder, one of several actions can be taken. These actions are specified by the **OnFileExistsInFailureArchive** parameter. The following table lists the available options:

Parameter	Indicates …
Overwrite	The new file will be stored in the failure archive folder and will overwrite the file that is already stored there.
OverwriteWith Warning	The new file will be stored in the failure archive folder and will overwrite the file that is already stored there. In addition, a warning will be written to the Error log file.
Fail	The new file will not be stored in the failure archive folder and an error will be written in the process file.
UserExit	The new file will not be stored in the failure archive folder and a user exit called **OnFileExistsInArchiveFolder** will be called automatically.
UserExitWith Overwrite	A user exit called **OnFileExistsInArchiveFolder** will be called automatically and then the file in the failure archive folder will be overwritten.

Scheduling Processes

The **ProcessScheduling** parameter enables you to determine whether the Distribution and Collection Agent will start the process automatically at regular intervals according to a predetermined period of time, or whether an external catalyst, such as an external scheduler, will activate the process.

The following table lists the available options for this parameter:

Parameter	Indicates …
Interval	The transfer specified in the process will be carried out automatically at predetermined intervals, according to the number of seconds specified in the **SchedulingInterval** parameter. The RestrictSchedulingIntervalTime parameter determines whether or not the process activation will be restricted to a specific time period. If so, the Scheduling FromTime and SchedulingToTime parameters specify the times between which the process will be activated.
ExternalActivation	The transfer specified in the process will only be carried out after an external catalyst begins the process, such as the DcaCLI.

Selecting Source Files

The **ProcessRecursive** parameter determines whether or not the contents of subfolders of the specified source folders will be included in the transfer. This enables you to specify a parent folder in the rule section of the process, rather than every folder that appears under it.

The following table lists the available options for this parameter:

Parameter	Indicates …
True	The contents of the subfolders of the source folder specified in the rule will be transferred to the destination location during the process.
False	Only the contents of the source folder specified in the rule will be transferred to the destination location during the process.

In addition, a file pattern can be specified in the **FileSelectionPattern** parameter to indicate which files to transfer during a process. This pattern is a string that may contain either of the following options:

Parameter	Indicates …
*	Any string of characters.
?	Any character.

When the source port is a Vault port, you can also transfer files with specific file categories. The **FileSelectionCategories** parameter enables you to list the file categories and their values, in the following format:

```
<FileSelectionCategories>
   <Category1>Value</Category1>
   <Category2>Value</Category2>
   <Category3>Value</Category3>
   …
</FileSelectionCategories>
```

The **FileSelectionRelationshipBetweenCategories** parameter defines whether at least one file category or all file categories listed in the **FileSelectionCategories** parameter must be defined in files that will be transferred.

Parameter	Indicates …
AND	Files in which all the file categories listed in the **FileSelectionCategories** parameter are defined will be transferred.
OR	Files in which at least one of the file categories listed in the **FileSelectionCategories** parameter is defined will be transferred. This is the default.

The following example shows how the parameters explained in this chapter can be used in the general process attributes of a process.

```
<Process>
   <ProcessName>MyProcess</ProcessName>
   <ProcessEnabled>True</ProcessEnabled>
   <ProcessType>N2N</ProcessType>
   <NonDuplicationMethod>Delete</NonDuplicationMethod>
   <OnFileExistsInDest>Overwrite</OnFileExistsInDest>
   <ProcessScheduling>Interval</ProcessScheduling>
   <SchedulingInterval>60</SchedulingInterval>
   <RestrictSchedulingIntervalTime>True</RestrictSchedulingIntervalTime>
   <SchedulingFromTime>03:00</SchedulingFromTime>
   <SchedulingToTime>05:00</SchedulingToTime>
   <ProcessRecursive>True</ProcessRecursive>
   <FileSelectionPattern>*.doc</FileSelectionPattern>
   <FileSelectionCategories>
      <Classification>Top Secret</Classification>
      <Subject>Salaries</Subject>
   </FileSelectionCategories>
   <FileSelectionRelationshipBetweenCategories>AND</FileSelection
    RelationshipBetweenCategories>
.
.
.
</Process>
```

In the above example, the process is called ‘MyProcess’. This process is enabled and is an N2N type process. The exact source and destination locations of the files to be transferred are specified in the rules section, which is not shown here. This process will run according to the ‘ProcessScheduling’ parameter, which specifies that a transfer will be activated every 60 seconds between 3.00 am and 5.00 am.

This example also indicates that if files are transferred and there is already an existing file with the same name in the destination location, the file being transferred will overwrite the file that already exists there. All files with the ‘.doc’ extension that are in the folder specified in the rule section or its subfolders (recursively) will be transferred to the specified destination location.

Finally, this example specifies that only files in which the 'Classification' file category is defined with the 'Top Secret' value and the 'Subject' file category is defined with the 'Salaries' value will be transferred.

Locking Source Files

The **LockOnRetrieve** parameter determines whether or not to lock the source file during the file transfer operation, in a Vault-to-Vault (V2V) or Vault-to-File System (V2FS) transfer.

The following table lists the available options for this parameter:

Parameter	Indicates …
True	The source files will be locked to other users during the file transfer operation.
False	The source files will not be locked to other users during the file transfer operation.

The **LockOnFS** parameter determines whether or not to lock the source file during the file transfer operation, in a File System-to-Vault (FS2V) transfer, and what to do if the source file cannot be locked.

The following table lists the available options for this parameter:

Parameter	Indicates …
None	If the source files on the File System cannot be locked, no action will be taken.
Fail	Any source files that cannot be locked will not be transferred, and a warning will be written to the Error log.
Skip	Any source files that cannot be locked will not be transferred.

The **LockOnFSAccessMode** parameter determines which access mode will be used for locking source files in a File System-to-Vault (FS2V) transfer. This parameter is relevant if the **LockOnFS** parameter is set to **Skip** or **Fail**.

For more information about the values for this parameter, see the values for the dwDesiredAccess parameter of the CreateFile function in the Windows Platform SDK.

The **LockOnFSShareMode** parameter determines which share mode will be used for locking source files in a File System-to-Vault (FS2V) transfer. This parameter is relevant if the **LockOnFS** parameter is set to **Skip** or **Fail**.

For more information about the values for this parameter, see the values for the dwShareMode parameter of the CreateFile function in the Windows Platform SDK.

Transfer Rules

The transfer rules section includes a list of one or more rules that define the source and destination locations of the files to transfer. The source and destination are called ports, and can be either a File System or a Vault.

The following example shows the generic structure of a rule in a process definition file.

```
<Rule1>
       <RuleName>NameoftheRule</RuleName>
       <SourcePort>
              .
              Port section
              .
       </SourcePort>
       <DestPort>
              .
              Port section
              .
       </DestPort>
</Rule1>
```

File System Port

When the port is a file system, the following parameters specify the location:

- **Name** – Specifies the logical name of the file system port.
- **Type** – Indicates that the port is a file system.
- **Folder** – The name of the folder where the files will be transferred to or from.

The following example shows how these parameters will appear in the process file.

```
<Name>MyFileSystem</Name>
<Type>FileSystem</Type>
<FolderName>c:\FileTransfer\UploadToSafe</FolderName>
```

In the above example, the name of the file system port is **MyFileSystem**. The file transfer will be carried out from a **file system**, from the **c:\FileTransfer\UploadToSafe** folder.

Note: In the Folder parameter, you can specify either a folder on the local machine or a folder on the network. If you specify a folder on the network, use the UNC convention (\\computername\sharedfolder\resource) and make sure that the user who is used to run the DCA service is a domain user with the appropriate access rights and authorizations on the file system network resource.

Vault Port

When the port is a Vault, the following parameters specify the location:

- **Name** – Specifies the name of the logical name of the Vault port.
- **Type** – Indicates that the port is a Vault.
- **VaultName** – The name of the source or destination Vault from/to where the files will be transferred.
- **SafeName** – The name of the Safe where the files are stored.
- **Folder** – The name of the folder where the files will be transferred to or from.
- **UserName** – The name of the Vault user who will authenticate to the Vault and access the files to transfer.

The following example shows how these parameters will appear in the process file.

```
<Name>CompanyVault</Name>
<Type>Vault</Type>
<VaultName>CompanyVault<VaultName>
<SafeName>VaultSafe<SafeName>
<FolderName>Root</FolderName>
<UserName>DCAUser1<UserName>
```

In the above example, the name of the Vault port is **CompanyVault**. The file transfer will be carried out to a **Vault** called **CompanyVault**. The files will be transferred to the **Root** folder of a Safe called **VaultSafe** by a user called **DCAUser1**.

Specifying a Vault

The Vault specified in the VaultName parameter indicates the Vault that files will be transferred to or from. A Vault parameter file contains all the parameters of this Vault and must be saved in the Vaults subfolder of the DCA installation folder.

To Specify a Vault for a File Transfer

1. In the DCA\Vaults folder, create a new subfolder to store the Vault parameter file. The name of the subfolder must be the same as the name of the Vault that will be specified by the Vault parameter file.
2. From the DCA\Samples folder, copy the Vault.ini file to the folder you created.
3. Either leave the Vault filename as Vault.ini, or rename it to indicate the name of the Vault that it represents, i.e., <*VaultName*>.ini.
4. In the Vault parameter file, specify the Vault parameters so that the Distribution and Collection Agent will be able to access it.
5. In the Process configuration file, in the SourcePort or DestinationPort section, specify the name of the Vault in the VaultName parameter. The VaultName parameter must be the same as the name of the Vault folder and the <*VaultName*>.ini file.

Specifying a User

The user name specified in the UserName parameter indicates the user that the Distribution and Collection Agent will use to access the Vault and carry out the file transfer. The user's logon credentials are stored in a credentials file that contains the user's Vault username and encrypted password. The name of the file indicates whose user credentials it contains. For example, DCAuser1.cred indicates that this credentials file contains the Vault username and password for a user called **DCAuser1**. The DCA changes this user's password every time it logs on to the Vault, and automatically updates the user's credentials file.

All the credential files for each Vault are stored under the DCA\Vaults folder in a subfolder for the Vault that these credential files can be used to access. For example, user credential files that are stored in the DCA\Vaults\ CompanyVault folder can be used to access a Vault called **CompanyVault**.

1. If the credentials file folder is DCA\Vaults, use the folder created above in *Specifying a Vault*,

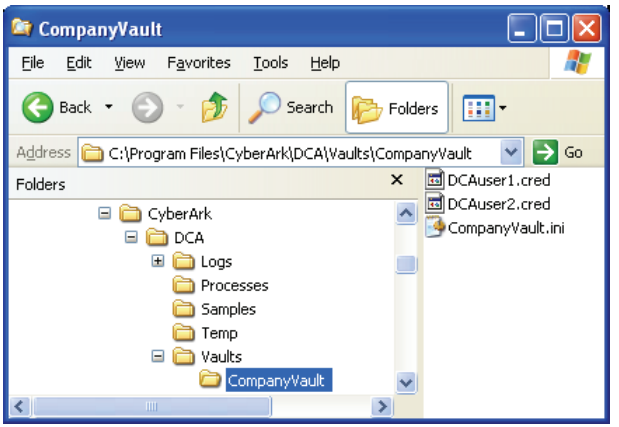
   or,

   Create a new subfolder to store the user credentials file.

   Note: The name of the subfolder must be the same as the name of the Vault that the credentials files will access.

2. Use the CreateCredFile utility to create a user credentials file for the user and store the credentials file in the appropriate Vault subfolder.
3. In the Process configuration file, in the SourcePort or DestinationPort section, in the UserName parameter, specify the name of the Vault user.

The following example shows the folder structure and where the Vault parameter file and the user credentials files will be stored so that the Distribution and Collection Agent can access them to carry out file transfers.



Transferring Files between Ports

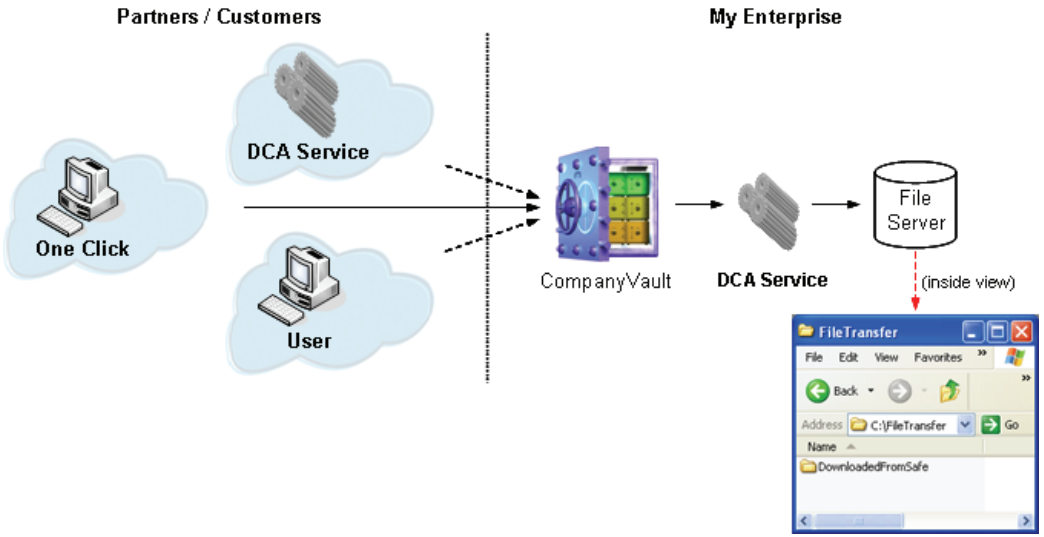
Transferring files from a file system to a Vault

The following example shows a process rule, called **MyRule**. The name of the source port is **MyFileSystem** and the files to be transferred will be taken from **c:\FileTransfer\UploadToSafe** in the local **FileSystem**. The files will be transferred to a destination port called Company Vault. A Vault user called **DCAuser** will access a Vault called **CompanyVault** and store the files in the **Root** folder of a Safe called **MySafe**.

```
<Rule1>
        <RuleName>MyRule</RuleName>
        <SourcePort>
              <Name>MyFileSystem</Name>
              <Type>FileSystem</Type>
              <FolderName>c:\FileTransfer\UploadToSafe</FolderName>
        </SourcePort>
       <DestPort>
              <Name>CompanyVault</Name>
              <Type>Vault</Type>
              <VaultName>CompanyVault</VaultName>
              <UserName>DCAuser</UserName>
              <SafeName>MySafe</SafeName>
              <FolderName>Root</FolderName>
       <DestPort>
</Rule1>
```

Transferring files from a Vault to a file system

The following example shows a process rule, called **MyRule**. The name of the source port is **CompanyVault** and the files to be transferred will be taken from a Vault called **CompanyVault**. A Vault user called **DCAuser** will be used to enter the Vault and retrieve the files to transfer from the **Root** folder in a Safe called **MySafe**. The files will be transferred to a destination port called **MyFileSystem** which is in a **FileSystem**, and is located at **c:\FileTransfer\DownloadedFromSafe**.



In order to carry out the file transfer in the diagram above, the following rule would be used:

```
<Rule1>
         <RuleName>MyRule</RuleName>
         <SourcePort>
                 <Name>CompanyVault</Name>
                 <Type>Vault</Type>
                 <VaultName>CompanyVault</VaultName>
                 <UserName>DCAuser</UserName>
                 <SafeName>MySafe</SafeName>
                 <FolderName>Root</FolderName>
         </SourcePort>
        <DestPort>
                 <Name>MyFileSystem</Name>
                 <Type>FileSystem</Type>
                 <FolderName>c:\FileTransfer\DownloadedFromSafe</FolderName>
        <DestPort>
</Rule1>
```

Transferring files from one Vault to another

The following example shows a process rule, called **MyV2VRule**. The name of the source port is **CompanyVault1** and the files to be transferred will be taken from a **Vault** called **CompanyVault1**. A Vault user called **DCAuser** will be used to enter the Vault and retrieve the files to transfer from the **Root** folder in a Safe called **Vault1Safe**. The files will be transferred to a **Vault** destination port called **CompanyVault2**. The destination Vault, also called **CompanyVault2**, will be accessed by a Vault user called **DCAuser**, who will store the transferred files in the **Root** folder of a Safe called **Vault2Safe**.

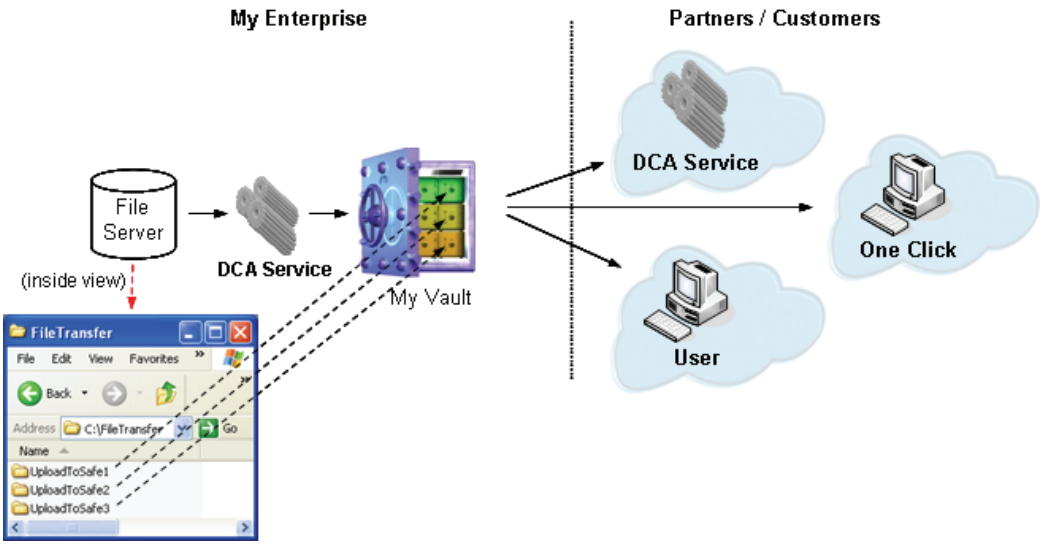
```
<Rule1>
         <RuleName>MyV2VRule</RuleName>
         <SourcePort>
                 <Name>CompanyVault1</Name>
                 <Type>Vault</Type>
                 <VaultName>CompanyVault1</VaultName>
                 <UserName>DCAuser</UserName>
                 <SafeName>Vault1Safe</SafeName>
                 <FolderName>Root</FolderName>
         </SourcePort>
        <DestPort>
                 <Name>CompanyVault2</Name>
                 <Type>Vault</Type>
                 <VaultName>CompanyVault2</VaultName>
                 <UserName>DCAuser</UserName>
                 <SafeName>Vault2Safe</SafeName>
                 <FolderName>Root</FolderName>
        <DestPort>
</Rule1>
```

Transferring files from a file server to multiple Safes

The following example shows a process in which files are copied from a file server to three Safes in a Vault. This process requires three rules:

- From C:\FileTransfer\UploadToSafe1 → MyVault Safe: UploadSafe1
- From C:\FileTransfer\UploadToSafe2 → MyVault Safe: UploadSafe2
- From C:\FileTransfer\UploadToSafe3 → MyVault Safe: UploadSafe3

For this example, the same user will carry out each rule.



The process in this example is called **MyProcess** and it is an **N2N** type transfer, so the specific source and destination locations will be specified in each rule. Only files with a ***.doc** extension will be transferred, and those that have been transferred successfully will be deleted from the source location.

Each rule specifies its own name, source and destination locations, and the name of the Vault user to use.

The above diagram requires the following rules:

```
<Process>
       .
       .
       .
       <Rule1>
          <Name>MyFileSystem1</Name>
          <RuleName>MyRule1</RuleName>
          <SourcePort>
              <Type>FileSystem</Type>
              <FolderName>c:\FileTransfer\UploadToSafe1</FolderName>
          </SourcePort>
          <DestPort>
              <Name>MyVault1</Name>
              <Type>Vault</Type>
              <VaultName>CompanyVault</VaultName>
              <UserName>DCAuser</UserName>
              <SafeName>UploadSafe1</SafeName>
              <FolderName>Root</FolderName>
          <DestPort>
       </Rule1>
       <Rule2>
          <Name>MyFileSystem2</Name>
          <RuleName>MyRule2</RuleName>
          <SourcePort>
              <Type>FileSystem</Type>
              <FolderName>c:\FileTransfer\UploadToSafe2</FolderName>
          </SourcePort>
          <DestPort>
              <Name>MyVault2</Name>
              <Type>Vault</Type>
              <VaultName>CompanyVault</VaultName>
              <UserName>DCAuser</UserName>
              <SafeName>UploadSafe2</SafeName>
              <FolderName>Root</FolderName>
       </Rule2>
```

```
        <Rule3>
           <RuleName>MyRule3</RuleName>
           <SourcePort>
                <Name>MyFileSystem3</Name>
                <Type>FileSystem</Type>
                <FolderName>c:\FileTransfer\UploadToSafe3</FolderName>
           </SourcePort>
           <DestPort>
                <Name>MyVault3</Name>
                <Type>Vault</Type>
                <VaultName>CompanyVault</VaultName>
                <UserName>DCAuser</UserName>
                <SafeName>UploadSafe3</SafeName>
                <FolderName>Root</FolderName>
        </Rule3>
        .
        .
        .
</Process>
```

Creating Rules Dynamically

Transfer Rules can be created dynamically, based on a definition of source and destination ports in the process definition file and the folder structure in the source port. In this way, entire folders can be copied to a destination with a single rule. A process whose rules are created this way is called an Automap process.

An automap process defines the master source port and the master destination port. When the process runs, a rule is created automatically for each folder under the master source port to copy each one to a folder under the master destination port.

An automap file transfer can be carried out for each type of file transfer that is supported by the Distribution and Collection Agent, specifically:

- File server → Vault
- Vault → File server
- Vault → Vault

Specifying Dynamic Rules

The automap process is defined in a process xml file, which contains all the process parameters.

General Process Attributes

The following parameters specify the automap process attributes:

- **ProcessType** – This parameter indicates that this is an automap process.
- **AutomapPattern** – This parameter defines the folders in the master source port that will be mapped during the process.

The following example indicates that the process is an automap process and that every folder under the folder specified in the master source port will be copied to the destination port.

```
<ProcessType>Automap</ProcessType>
<AutomapPattern>*</AutomapPattern>
```

Rules

In an automap process, the rules are defined dynamically. Therefore, no rules are specified and this section is left empty.

Master Source Port and Master Destination Port

The master source and destination ports can be either a file system port or a Vault port. Their attributes are the regular file system or Vault ports' attributes described earlier in *Transfer Rules*.

The following example shows the generic structure for an automap process definition file.

```
<Process>
       <ProcessName>MyAutomapProcess</ProcessName>
       <ProcessType>Automap</ProcessType>
       <AutomapPattern>*</AutomapPattern>

<MasterSourcePort>
       .
       Source port attributes
       .
</MasterSourcePort>

<MasterDestPort>
       .
       Destination port attributes
       .
</MasterDestPort>

</Process>
```

The master source port definition must specify all the port attributes.

As the process rules are created dynamically, the master destination port does not need to specify all the port attributes. However, the missing attributes affect how the rules are created. This is described further in the next section.

Automap Process Operation

During an automap process, the process looks for subfolders of the source folder defined in the master source port, and builds a transfer rule for each subfolder. The destination of each rule depends on the destination port type and attributes:

Mapping to a File System Destination Port

When the master destination port is a file system, each subfolder of the source folder specified in the master source port is mapped to a corresponding folder under the destination folder specified in the master destination port.

The following example shows the port parameters for a process in which all the subfolders of the **Transfer** folder in a Safe called **CompanySafe** will be copied by a Vault user called **DCAuser** to corresponding folders under the c:\downloads folder on a File System.

```
<Process>
   .
   .
   .
   <MasterSourcePort>
      <Name>CompanyVault</Name>
      <Type>Vault</Type>
      <VaultName>CompanyVault</VaultName>
      <UserName>DCAuser</UserName>
      <SafeName>CompanySafe</SafeName>
      <FolderName>Root\Transfer</FolderName>
   </MasterSourcePort >

   <MasterDestPort>
      <Name>IncomingFolders</Name>
      <Type>FileSystem</Type>
      <FolderName>c:\FileTransfer\Incoming</FolderName>
   </MasterDestPort>
   .
   .
   .
</Process>
```

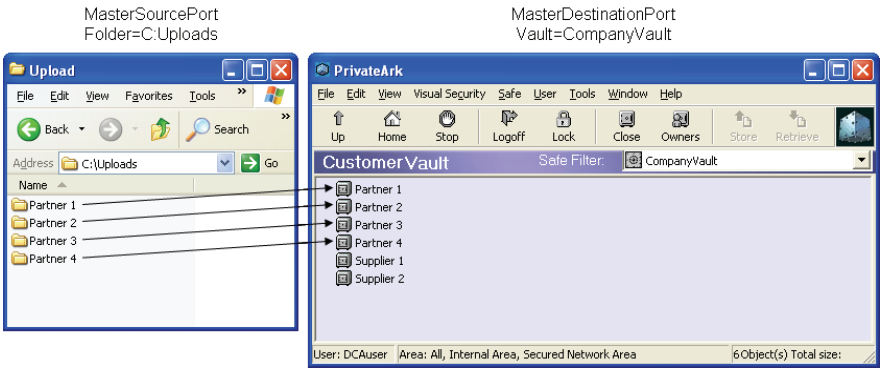
Mapping to a Vault Destination Port

When the master destination port is a Vault, the mapping depends on the port attributes that are specified in the process file.

- **Mapping to Safes**

  When the destination Safe is not specified, files are automatically transferred to Safes with the same name as the folders whose contents are being transferred. Files will be transferred to the Root folder in the Safe.

  In the following example, the contents of each subfolder of **C:\Uploads** will be copied to a corresponding Safe in a Vault called **CompanyVault** by a Vault user called **DCAuser**.



The above example requires the following parameters in the process file:

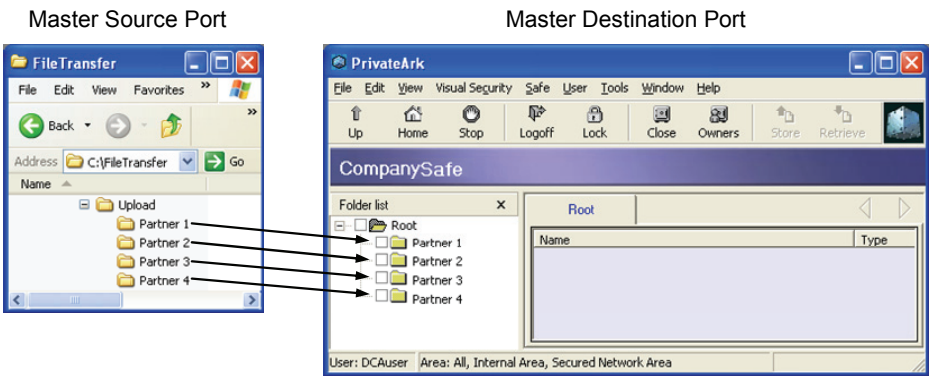
```
<MasterSourcePort>
   <Name>OutgoingFolders</Name>
   <Type>FileSystem</Type>
   <FolderName>C:\Uploads</FolderName>
</MasterSourcePort>

<MasterDestPort>
   <Name>CompanyVault</Name>
   <Type>Vault</Type>
   <VaultName>CompanyVault</VaultName>
   <UserName>DCAuser</UserName>
<MasterDestPort>
```

- **Mapping to folders under Root in a specific Safe**

  When the name of the destination folder is not specified, each subfolder of the source folder is mapped to a corresponding folder under the Root folder in the destination Safe.

  In the following example, the contents of each subfolder of **C:\FileTransfer\Upload** will be copied to corresponding subfolders of the Root folder in a Safe called **CustomerSafe** in a Vault called **CompanyVault** by a Vault user called **DCAuser**.



The above example requires the following parameters in the process file:

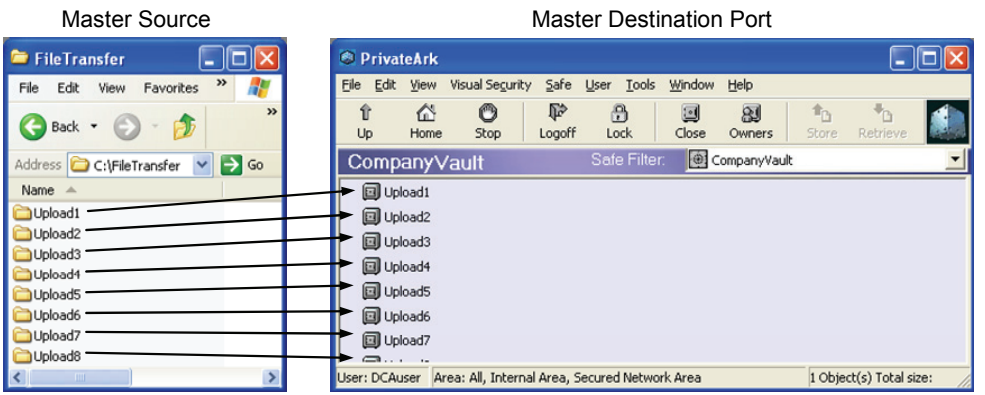
```
<MasterSourcePort>
   <Name>OutgoingFolders</Name>
   <Type>FileSystem</Type>
   <FolderName>c:\FileTransfer\Upload</FolderName>
</MasterSourcePort>

<MasterDestPort>
   <Name>CompanyVault</Name>
   <Type>Vault</Type>
   <VaultName>CompanyVault</VaultName>
   <UserName>DCAuser</UserName>
   <SafeName>CompanySafe</SafeName>
<MasterDestPort>
```

Distributing Data to a Dynamic Number of Customers

The following example shows a data transfer process in which multiple rules are carried out. These rules include data transfers to a growing number of customers. The Distribution and Collection Agent enables you to create these rules automatically by defining an automap process.

In the following example, the process will upload files from the **C:\FileTransfer\Uploads<#>** folder in the local file system to a Vault called **CompanyVault** using a Vault user called **DCAuser** Automatic rules will be built to transfer the contents of each folder into a corresponding Safe.



The above diagram requires the following process XML file:

```
<Process>
   <ProcessName>MyAutomapProcess</ProcessName>
   <ProcessType>Automap</ProcessType>
   .
   .
   .
   <MasterSourcePort>
      <Name>OutgoingFolders</Name>
      <Type>FileSystem</Type>
      <FolderName>c:\FileTransfer</FolderName>
   </MasterSourcePort>

   <MasterDestPort>
      <Name>CompanyVault</Name>
      <Type>Vault</Type>
      <VaultName>CompanyVault</VaultName>
      <UserName>DCAuser</UserName>
   <MasterDestPort>
   .
   .
   .
</Process>
```

Any new customer folders that are added to C:\FileTransfer will automatically be transferred to a corresponding Safe in the Vault by this process.

User Exits

User exits enable users to customize the process carried out by the Distribution and Collection Agent, by calling a user-defined program during a file transfer. User exits can be activated by a process at various stages. This enables you to customize the standard file transfer process to your specific business logic.

User exits can also be activated under specific circumstances. For example, a user exit can be activated when an error occurs, and send an email notification to a particular user.

The following table lists the user exits that are supported by the Distribution and Collection Agent and when they can be activated:

Parameter	To be used …
PreProcess	Before a process instance starts running.
PostProcess	After a process instance has finished running.
PreRule	Before a transfer rule starts running.
PostRule	After a transfer rule has finished running.
PostFindFiles	After a transfer rule has located the files to transfer.
OnFileInit	When a file to be transferred is initialized.
PreFile	Before a file is transferred.
PostRetrieveFile	In a Vault-to-Vault transfer, after a file is retrieved from the source Vault.
OnFileExistsInDestination	If a file with the same name as the file being transferred already exists in the destination location and before the file is copied to the destination.
PostFile	After a file has been transferred.
OnError	When an error occurs.

For more details about the above user exits, refer to *User Exits*, page *118*, in *Appendix C: Process Definition Parameters*.

The Distribution and Collection Agent can activate user exits in the following languages/formats:

- Perl
- C++
- Batch files

Writing Perl User Exits

Perl user exits are functions that are written in a Perl script file and activated as part of a process run by the Distribution and Collection Agent. In order to use Perl user exits, Perl must be installed on the same machine as the Distribution and Collection Agent.

As part of the Distribution and Collection Agent installation, the Perl user exits feature installs a Perl package, FTPerlWrap.pm. This package enables you manipulate the Distribution and Collection Agent behavior from your Perl function. Use the 'use' Perl command to include this package in your Perl script.

When a process activates a Perl user exit, the process sends the user exit a special object as a parameter. The object type depends on the user exit type. Methods of this object can be called from a Perl function and, in this way, change the Distribution and Collection Agent process.

Note: By default, Perl user exits are configured not to run in parallel. However, this can be changed and they can run in parallel according to your requirements.

For a full list of Perl user exit parameters, refer to *Specifying C++ User Exits in the Process File*, page *68*.

Each Perl function issues a return code which indicates to the Distribution and Collection Agent whether or not the user exit finished successfully.

For a full list of user exit return codes, refer to *Return Codes for User Exits*, page *76*.

The following example shows a Perl function that will be used as a PreFile user exit.

The user exit receives an object from the Distribution and Collection Agent process that represents the file to transfer and saves it in a variable called $FTTransferFile. The user exit then calls methods on this object in order to change the destination file name. At the end of the function, it returns a zero to indicate that the user exit finished successfully.

```
use FTPerlWrap;

sub PreTransferFile
{
   my $FTTransferFile = $_[0];
   $DestFile = $FTTransferFile->GetWVaultDestFile();
   $DestFile->SetFileName("filename");

   return 0;
}
```

Notes:

- If you install the Perl user exits feature on the Distribution and Collection Agent machine where Perl is not installed, the installation will display an error message.
- If Perl or the Perl user exits feature is not installed, and a process with Perl user exits is run, the Distribution and Collection Agent will not be able to load the process, and an error message will be written in the service log.

Specifying Perl User Exits in the Process file

To specify a Perl user exit in the process xml file, define the following parameters.

Parameter	Description	Required
UserExitType	The type of user exit.	Yes
UserExitFileName	The file name of the user exit.	Yes
UserExitFuncName	If UserExitType=Perl, specify the Perl function name. If this parameter is not specified, a function with the same name as the user exit will be executed.	No

The following example defines a PreFile Perl user exit. The Perl script file name is **c:\userexits\prefile.pl** and the Perl function name is **OnPreFile**.

```
<UserExits>
   <PreFile>
      <PreFile1>
         <UserExitType>Perl</UserExitType>
         <UserExitFileName>c:\userexits\prefile.pl</UserExitFileName>
         <UserExitFuncName>OnPreFile</UserExitFuncName>
      </PreFile1>
   </PreFile>
</UserExits>
```

Writing C++ User Exits

C++ user exits are C++ functions that are activated as part of a Distribution and Collection Agent process. The C++ functions are defined within a dynamic link library (dll), which is loaded by the Distribution and Collection Agent when the process starts.

The Distribution and Collection Agent installation supplies a header file that defines a C++ class called **IFTUserExitsHandler**. This class contains a method for each user exit type with an empty implementation.

To implement C++ user exits, write your own C++ class that inherits from the IFTUserExitsHandler class. In the class you create, you can implement the methods of the user exits that will meet your needs. This implementation will override the empty implementation exits in the basic IFTUserExitsHandler class.

Like Perl user exits, C++ user exits receive objects as parameters from the Distribution and Collection Agent process and they must have a return code. The parameter is given as a regular C++ method parameter and the return code is the method return code.

For a full list of C++ user exit parameters, refer to *Specifying C++ User Exits in the Process File*, page *68*.

For a full list of user exit return codes, refer to *Return Codes for User Exits*, page *76*.

In order for the Distribution and Collection Agent to be able to load your dll, you must define a function called **UserExitsDllEntryFunc** in the dll. This function must be exported by the dll and must return a pointer to an instance of your C++ class. The function declaration is displayed below:

```
__declspec(dllexport) IFTUserExitsHandler* UserExitsDllEntryFunc();
```

The following C++ code defines a class called **MyUserExitHandler**. This class inherits the basic class IFTUserExitHandler and implements the PreRule method.

```
// UserExitDllSample.cpp

#include "stdio.h"
#include "stdafx.h"
#include "IFTUserExitsHandler.h"
#include "CFTWTransferRule.h"
#include "CFTWTransferInfo.h"

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved)
{
    return TRUE;
}

//-------------------------------------------
// user exits handler class
//-------------------------------------------
class MyUserExitsHanlder : public IFTUserExitsHandler
{
   virtual long PreRule(CFTWTransferRule* pWTransferRule);
   {
      int retries = pWTransferRule->GetTransferInfo()->GetRuleRetriesNum();
      return 0;
   }
}

MyUserExitsHanlder UserExitsHanlder;

//-------------------------------------------
// dll entry function
//-------------------------------------------
__declspec(dllexport)
IFTUserExitsHandler* UserExitsDllEntryFunc()
{
   return &UserExitsHanlder;
}
```

In the above example, the PreRule method accepts an object that represents the transfer rule as a parameter. The method uses this object to retrieve the RuleRetries parameter defined for this rule and returns zero as a return code. At the end of the sample, you can see the declaration of the UserExitsDllEntryFunc function exported by the dll.

Specifying C++ User Exits in the Process File

To specify a C++ user exit in the process xml file, define the following parameters in the general process attributes in the process file.

Parameter	Description
UserExitsDlls	The full pathnames of dlls for C++ user exits.
UserExitDll<#>	The full pathname of each user exit dll.

Specify the following parameters in the user exits attributes of the process file.

Parameter	Description	Required
UserExitType	The type of user exit.	Yes
UserExitDllReference	If UserExitType=dll, specify the reference to the full pathname of the dll.	Yes

The following example defines two C++ PostRule user exits. The first user exit is implemented in **c:\UserExitDllSample1.dll** and the second user exit is implemented in **c:\UserExitDllSample2.dll**.

```
<UserExitsDlls>
   <UserExitDll1>c:\UserExitDllSample1.dll</UserExitDll1>
   <UserExitDll2>c:\UserExitDllSample2.dll</UserExitDll2>
</UserExitsDlls>
.
.
.
<UserExits>
   <PostRule>
      <PostRule1>
         <UserExitType>Dll</UserExitType>
         <UserExitDllReference>UserExitDll1</UserExitDllReference>
      </PostRule1>
      <PostRule2>
         <UserExitType>Dll</UserExitType>
         <UserExitDllReference>UserExitDll2</UserExitDllReference>
      </PostRule2>
   </PostRule>
</UserExits>
```

Perl and C++ User Exits Parameters

When activating Perl or C++ user exits, the Distribution and Collection Agent process passes the user exit an object that can be used to retrieve information about the process or change the process behavior.

The type of object that is transferred to the user exit depends on two factors:

- **User exit type** – A PreProcess user exit will receive a "process object" that can be used to acquire global information about the transfer process. A PreFile user exit will receive a "file object" that can be used to acquire information or change the details of a specific file that is being transferred.

- **Transfer type** – Different transfer types require the objects that they receive to carry out different activities. For example, in a Vault-to-Vault transfer, files are transferred while encrypted. In the user exit, you can change this and decrypt the files while they are transferred. Only the object that is given to a file level user exit in a V2V transfer should export this type of functionality.

The following table describes the objects that are passed to a Perl or C++ user exit according to the user exit type and transfer type.

Transfer type / User exit type	V2V	V2FS	FS2V
Process level: PreProcess PostProcess	CFTWProcess Instance	CFTWProcess Instance	CFTWProcess Instance
Rule level: PreRule PostRule PostFindFiles	CFTWV2VTransfer Rule	CFTWV2FS TransferRule	CFTWFS2V TransferRule
File level: OnFileInit PreFile PostRetrieveFile OnFileExistsInDestination PostFile	CFTWV2V TransferFile	CFTWV2FS TransferFile	CFTWFS2V TransferFile
OnError	CFTWErrorContext	CFTWErrorContext	CFTWErrorContext

The header files that contain the definitions of the above objects are created during installation in the DCA\API folder.

Writing Batch User Exits

Batch user exits are batch scripts that are activated as part of a Distribution and Collection Agent process. The batch script receives parameters from the Distribution and Collection Agent as read-only strings. These parameters give information about the current process, such as the details of the file that is being transferred. The parameters that are sent by the Distribution and Collection Agent depend on the user exit type. For example, the PreProcess user exit receives different parameters to those received by the PreFile user exit.

For a full list of Batch user exit parameters, refer to *Batch User Exits Parameters*, page *72*.

Like Perl and C++ user exits, Batch user exits also issue a return code that informs the Distribution and Collection Agent process whether or not the user exit finished successfully.

For a full list of user exit return codes, refer to *Return Codes for User Exits*, page *76*.

User exits that are run at File and Rule level can interact with a source or destination Vault using PACLI.

For more information about how batch user exits interact with source and destination Vaults using PACLI, refer to *Using PACLI in Batch User Exits*, page *75*.

The following example displays a user exit that runs at Rule level (PreRule, PostRule, PostFindFiles) in a Vault-to-Vault transfer. It prints the parameters sent by the Distribution and Collection Agent process to the c:\log.txt file.

```
echo In Batch user exit > c:\log.txt
echo rule level parameters: %1, %2, %3, %4, %5, %6, %7, %8 > c:\log.txt
```

A list of the parameters appears in the table below, in *Batch User Exits Parameters*.

Batch User Exits Parameters

When activating Batch user exits, the Distribution and Collection Agent process transfers a list of strings to the user exit. The user exit type and the transfer type determine which strings are passed to the user exit.

The following table describes the order and values of the strings sent to each user exit:

Process level user exits:	
User exits:	PreProcess PostProcess
Strings:	process type process name process scheduling method scheduling interval / external activation level (depending on the scheduling method)
Rule level user exits:	
User exits:	PreRule PostRule PostFindFiles
Strings:	rule type rule name source vault name (when the source port is a Vault) source safe name (when the source port is a Vault) source folder name destination vault name (when the destination port is a Vault) destination safe name (when the destination port is a Vault) destination folder name

File level user exits:	
User exits:	OnFileInit PreFile PostFile PostRetrieveFile OnFileExistsInDestination
Strings:	transfer file type source vault name (only when source port is a Vault) source safe name (only when source port is a Vault) source folder name source file name destination vault name (only when destination port is a Vault) destination safe name (only when destination port is a Vault) destination folder name destination file name
OnError user exit:	
Strings:	object type (e.g., "CFTWV2FSTransferFile") phase type (e.g., "PreOperations") error type (error/failure) last error message last error code source details for vault peer: "vault\safe\folder", for vault file: "\vault\safe\folder\path", for FS peer: "folder-path", for FS file: "file-path" destination details (like source details) user exit return code (if the error occurred in a user exit, you get here the error code of the user exit).

Specifying Batch User Exits in the Process file

To specify a batch user exit in the process xml file, define the following parameters.

Parameter	Description	Required
UserExitType	The type of user exit.	Yes
UserExitFileName	The file name of the user exit.	Yes
UserExitUsesVaultSession	If UserExitType=batch, specify whether or not the batch user exit will use a session to access the source or destination Vaults.	Yes

The following example defines a PostRetrieveFile batch user exit. The batch file name is **c:\userexits\PostRet.cmd** and the user exit does not require access to a Vault.

```
<UserExits>
   <PostRetrieveFile>
      <PostRetrieveFile1>
         <UserExitType>Batch</UserExitType>
         <UserExitFileName> c:\userexits\PostRet.cmd</UserExitFileName>
         <UserExitUsesVaultSession>False</UserExitUsesVaultSession>
      </PostRetrieveFile1>
   </PostRetrieveFile>
</UserExits>
```

Using PACLI in Batch User Exits

User exits at file and rule levels, including OnError user exits relating to errors that occurred at these levels, can interact with the source and destination Vaults using the PrivateArk Command Line Interface (PACLI).

To use PACLI from a batch user exit, set the **UserExitUsesVaultSession** parameter to **True** in the user exit definition, as follows:

```
<PreFile1>
   <UserExitType>Batch</UserExitType>
   <UserExitFileName>c:\userexit.cmd</UserExitFileName>
   <UserExitUsesVaultSession>True</UserExitUsesVaultSession>
</PreFile1>
```

After that, you may use PACLI commands in your batch file using the following guidelines:

- Do not log onto the Vault from PACLI. Your PACLI session is already logged on using the user that is used by the DCA for the current rule or file transfer.
- At the end of the batch file, you must call "PACLI term", regardless of whether or not you called any PACLI commands.
- In V2V file transfers, both sessions will be applied to the PACLI session. The batch file must define the appropriate default Vault and default user for each PACLI command.
- When the **UserExitUsesVaultSession** parameter is set to **False**, the DCA does not apply a session to PACLI. In this case, you can also use PACLI commands in your batch file, but the PACLI session must be created as in a regular PACLI script.

Return Codes for User Exits

A Distribution and Collection Agent process that activates a Perl / C++ / Batch user exit requires a return code. This return code tells the Distribution and Collection Agent process how the user exit ended:

Return code	Indicates …
0	The user exit was completed successfully.
> 0	The user exit was completed, but a warning was issued.
< 0	The user exit was not completed due to an error. (The Distribution and Collection Agent will retry.)
< -16,000	The user exit failed. (The Distribution and Collection Agent will not perform a retry.)

Manipulating a Transfer Destination

User exits influence the default behavior of the Distribution and Collection Agent, and enable you to customize file transfers.

Different types of user exits can be activated at various stages of a process, rule, or file transfer. It is important to identify exactly when the user exit is required in order to implement it correctly and maximize its effect.

The following example describes a scenario in which files will be transferred from a file system into a Vault, and their destination folders will be mapped automatically. However, in this scenario, the destination Safe name is not exactly the same as the source folder name. For example, whereas the name of the source folder is 'Customer<#>', the name of the destination Safe will be 'To_Customer<#>'.

This requires the name of the destination Safe to be changed for every rule that is created automatically. The correct user exit to use in this case is the PreRule user exit which you can activate before a rule is carried out.

The following Perl script is a user exit that will change destination Safe names to the required format (To_Customer<#>):

```
use FTPerlWrap;

sub OnPreRule
{
$RuleObject = $_[0];
$DestPort = $RuleObject->GetWVaultDestDataPeer();
$SafeName = $DestPort->GetSafeName();
$SafeName = "To_" . $SafeName;
$DestPort->SetSafeName($SafeName);
}
```

In the process file, there must be a reference to the Perl script file that contains this user exit. The above example requires the following process file:

```
<Process>
   <ProcessName>MonthlyBillsDistribution</ProcessName>
   <ProcessType>Automap</ProcessType>
   …
   <MasterSourcePort>
      <Type>FileSystem</Type>
      <FolderName>c:\uploads</FolderName>
   </MasterSourcePort>

   <MasterDestPort>
      <Type>Vault</Type>
      <VaultName>MyVault</VaultName>
   <MasterDestPort>

   <UserExits>
      <PreRule>
         <PreRule1>
            <UserExitType>Perl</UserExitType>
            <UserExitFileName>c:\userexits\prerule.pl</UserExitFileName>
            <UserExitFuncName>OnPreRule</UserExitFuncName>
         </PreRule1>
      </PreRule>
   </UserExits>

</Process>
```

Monitoring File Transfer Processes

Log files

The Distribution and Collection Agent manages the following log files to enable you to monitor file transfer processes:

- Service log
- Process log
- Error log
- Debug log
- Activity log

Service Log

The service log describes all main service activities. Each line in this log contains the date and time when the process was carried out, and a corresponding message.

The following list indicates the typical events that are documented in this log:

- Service initialization and shutdown, including the DCA version,
- Process and rule parameter definitions,
- Process startup, including the process name and instance id,
- Process termination, including the termination status (success, warning or failure) and the number of rules that were finished successfully, with warnings or failure,
- Loading or reloading of the Distribution and Collection Agent configuration file (dca.ini) and process configuration files,
- Errors that occur during Distribution and Collection Agent startup or while trying to load a process configuration file.

Process Log

The process log describes all the activities that are carried out by the process. A new process log is created for every process that is carried out.

Each line in this log contains the following information:

- Date
- Time
- Source port and destination port in the transfer
- Rule number of the rule carried out
- File number of the file being transferred
- A message that is relevant to the activity

An entry that is relevant to a specific rule will contain the source and destination ports and the rule number. An entry that is relevant to a specific file will contain the file number.

The following list indicates the typical events that are documented in this log:

- Process startup, including the process name and instance id,
- Process termination, including the termination status (success, warning or failure) and the number of rules that were finished successfully, with warnings or failure,
- Rule startup, including the source and destination folders,
- Rule termination, including the termination status (success, warning or failure) and the number of files that finished successfully, with warnings or failure,
- File transfer startup, including the source and destination full file names and transfer file id,
- File transfer termination, including the termination status (success, warning or failure),
- All activities carried out by the process, its rules and its file transfers,
- All errors that occurred during the process.

Error Log

A centralized error log file contains all the errors that occur while the Distribution and Collection Agent service is running. This log enables you to determine quickly if any errors occurred and its cause.

Each line in this log contains the following information:

- Date
- Time
- Source port and destination port in the transfer
- Rule number of the rule carried out
- File number of the file being transferred
- A message that is relevant to the activity

An entry that is relevant to a specific rule will contain the source and destination ports and the rule number. An entry that is relevant to a specific file will contain the file number.

Errors that occur during the Distribution and Collection Agent startup phase, before the error log is created, are written in the service log.

Debug Log

The debug log is an optional log that describes the details of all the activities carried out by the Distribution and Collection Agent.

To enable the debug log, in the DCA.ini, define the following parameter:

```
EnableDebugLog=Yes
```

The default value for this parameter is ‘No’.

Activity Log

The activity log describes activities that involve access to a Vault. This log file is created when the debug log is created.

Accessing Log Files on the Local Drive

All log files are saved under the logs folder that is created during the Distribution and Collection Agent installation. By default, this folder is created directly under the main installation directory and is called 'Logs'.

Log Recycling

The Distribution and Collection Agent can recycle the log files by time or by size. Old log files are saved in a subfolder of the Logs folder called 'old'. The date and time when the recycle took place is added to the file name. Each time the Distribution and Collection Agent starts, log files that exist are recycled, regardless of the time or size recycling limit. Log files of processes that will not be executed, will not be recycled.

To configure the Distribution and Collection Agent to recycle the log files after a specific number of minutes, in the DCA.ini, specify the following parameter:

```
LogRecycleIntervalMinutes=<minutes>
```

To configure the Distribution and Collection Agent to recycle the log files when a log file exceeds a specific limit, in the DCA.ini, specify the following parameter:

```
LogRecycleSizeMB=<MB>
```

If none of these parameters are specified, log files will be recycled automatically when they reach 25MB.

Deleting Old Log Files

Log files that have been moved to the Logs\old folder can be deleted periodically. The **PeriodicWorkIntervalSeconds** parameter in the DCA.ini determines how frequently the DCA will check the Logs\old folder for log files that should be deleted. Log files will be deleted if the number of days specified in the **KeepOldLogsDays** parameter has passed since they were last used.

For more information about the DCA.ini, refer to *Appendix A: Dca.ini*, page *104*.

Log Delimiter

A delimiter character separates the log fields in each line in the log file. This character improves log readability and enables you to transfer and analyze the contents of the log file to third party database applications, such as Microsoft Excel.

The default delimiter character is '|'. To define a different character, in the DCA.ini, specify the following parameter:

```
LogsColumnsDelimiter
```

Tracking File Transfer Processes

The DCA can be configured to monitor all file transfers that it carries out using File Level Tracking (FLT). This enables users to track files that have been transferred and to make sure they have reached their destination. Users can retrieve and query the status of transferred files with Cyber-Ark API functions, so that the DCA monitoring activities can be integrated into existing monitoring infrastructures.

Transfer Steps

File Level Tracking enables the user to track simple 'one step' transfers, as well as 'multiple step' transfers.

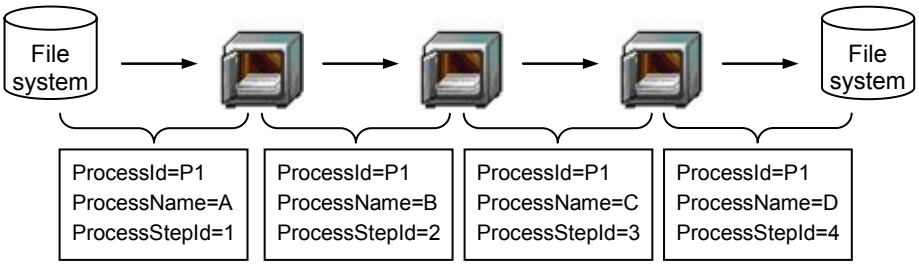
- One step transfer – This comprises one process that transfers files from one Safe to another (V2V), from a Safe to a File System (V2FS), or from a File System to a Safe (FS2V).
- Multiple step transfer – This comprises a group of processes, in which one process follows another. Initially, files are transferred to a specific location by one process, and are then transferred to a different location by another process. File Level Tracking enables users to track a file through each step, until it reaches its final destination.

In order to track files in a multiple step transfer, all the file transfer processes must share a common identifier to create a process group. This identifier must be specified in the **ProcessId** parameter in the process xml file. For more information about specifying the ProcessId parameter, refer to *Configuring File Level Tracking*, page *93*.

Each process within the group implements one step of the transfer. The step identification that is implemented by the process must be specified in the **ProcessStepId** parameter in the process xml file. For more information about

specifying the ProcessStepId parameter, refer to *Configuring File Level Tracking*, page *93*.

The following example shows a file transfer that consists of four processes. All the processes have the same Process identification (ProcessId). The process name identifies the specific process, and the Process step identification (ProcessStepId) indicates the step that is implemented by each process.



All the processes in the above path belong to a **ProcessId** called 'P1', although each individual process has its own **ProcessName**. In addition, each process is marked with a **ProcessStepId**, which indicates its step in the transfer.

Events

Each time a file is transferred, whether from Safe to Safe (V2V), from Safe to File System (V2FS), or from File System to Safe (FS2V), the DCA can be configured to store information about the transfer in a record called an Event.

In a multiple steps transfer, when a file is transferred several times by a number of processes, each step can be configured to create an Event that describes the transfer that was carried out. These Events enable users to track a file through all the steps in the transfer, from its origin to its final destination.

The **CreateFileEvents** parameter in the process xml file of the process that implements this step determines whether or not a step will create Events. For more information about specifying the CreateFileEvents parameter in the process xml file, refer to *Configuring File Level Tracking*, page *93*.

Creating Events

The events that are created by the DCA are stored in different locations according to the type of the transfer, as shown in the following table:

File Transfer Type	Describes
V2V	The DCA creates the Event in the source Safe of the transfer.
V2FS	The DCA creates the Event in the source Safe of the transfer.
FS2V	As there is no source Safe, the DCA creates the Event in the destination Safe of the transfer.

Types of Events

The DCA creates different types of Events according to the stage of the file transfer. The following table displays the different types of Events that can be created:

Event Type	Event Type ID	Describes
File transfer started	1	This Event is created when a file transfer starts to run.
File transfer succeeded	2	This Event is created when a file transfer ends successfully.
File transfer failed	3	This Event is created when a file transfer fails.

- When a file is transferred successfully, the following Event types will be created:
  File transfer started and **File transfer succeeded**.
- When a file transfer fails, the following Event types will be created:
  File transfer started and **File transfer failed**.

For more information about the contents of Events, refer to *Event Properties*, page *91*.

Consolidating Events

Every DCA process can be configured to create Events. The Events are stored in different locations, according to the type of the transfer (V2V, V2FS, FS2V).

In the following diagram, the DCA transfers the file from the Safe to the File System. The Events that describe this transfer will be stored in the source Safe.

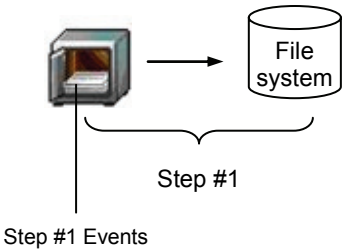


Figure 1: One step file transfer

In the next diagram, the DCA moves the file through a multiple steps transfer. The Events that describe each step will be stored in the relevant Safe for that step. This is either the source or destination Safe, according to the type of transfer that is carried out. For more information, refer to *Creating Events*, page *86*.

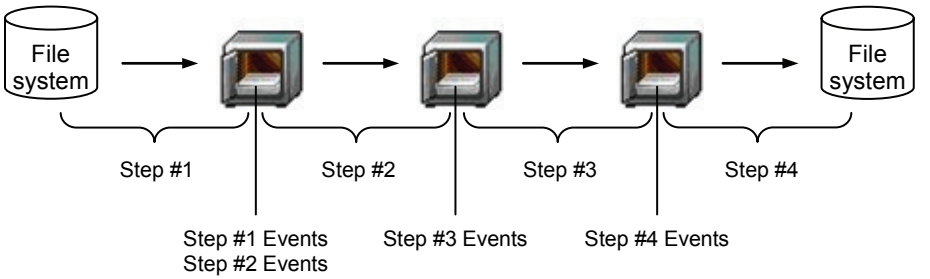
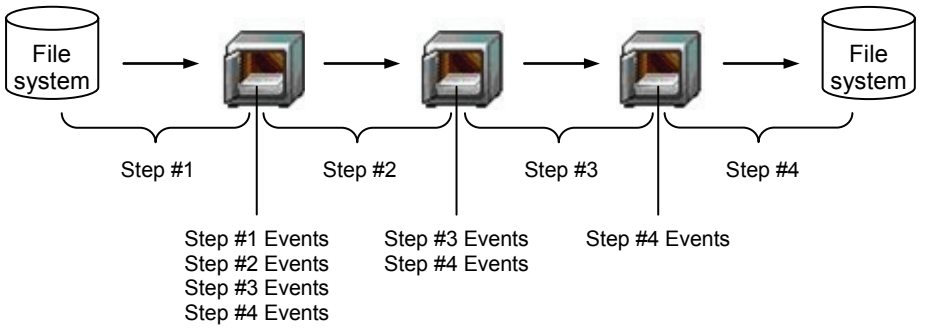


Figure 2: Multiple step file transfer

The Events that are created by step #1, an FS2V step, will be stored in the destination Safe of step #1. Events that are created by step #2, a V2V step, will be stored in the source Safe of step #2. This is the same safe where the Events of step #1 were stored. Events that are created by step #3, also a V2V step, will be stored in the source Safe of step #3. And finally, Events that are created by step #4, which is an FS2V step, will be stored in the source Safe of step #4.

In a multiple steps transfer that contains V2V steps, these steps can be configured to consolidate all the Events of a file transfer in a single location. At the end of the transfer, all the Events that were created during this transfer by all the steps that are part of it will be consolidated in one location in the first Safe used during the transfer, where the user can view them as one complete transfer. Each V2V step in the transfer must be configured to transfer Events back to the previous Safe in the transfer. As a result, all the Events will eventually be transferred to the first Safe used in the transfer where they will be consolidated.

The following diagram shows how Events created by each step of this transfer are consolidated in the first Safe of the transfer. As Step #2 and step #3 are configured to transfer Events, the Events that were created in the third Safe will also be stored in the second Safe, and all the Events stored in the second Safe, will also be stored in the first Safe, which is the destination Safe of step #1. After the Event have been consolidated, the user will be able to view the Events of the entire multiple step process.



The **TransferFileEvents** and **TransferEventsInterval** parameters in the process configuration file of the process that implements this step specify whether or not a step will transfer Events back to the previous Safe in the transfer. For more information about these parameters, refer to *Configuring File Level Tracking*, page *93*.

File Level Tracking Properties

In order to enable File level tracking, the DCA manages identification and tracking properties in the files it transfers. These details are added as File Categories on the files. These File Categories are built-in and must not be changed by the user.

File Descriptors

File descriptors are properties that describe and identify the file that is transferred. When the DCA uses file level tracking, these properties will be added to each file that is transferred.

File Descriptor	Indicates …	File Category
File ID	A unique string that identifies the file that is being transferred.	__FileID__
Process ID	A unique string that identifies the group of processes that comprise the complete file transfer. This is the ProcessId that is specified in the process xml file of the processes that carries out the transfer.	__ProcessID__
Process instance ID	A unique string that identifies the process instance that transferred this file. Each step in the file transfer is carried out by another process instance. This descriptor specifies the ID of the process instance in the first step.	__ProcessInstanceID__

In all types of transfers, if some of the file descriptors already exist in the file that is being transferred, the DCA will add those that are missing. Once the DCA has created a descriptor on a file, it will remain the same in all future transfers.

The **CreateFileDescriptors** parameter in the process xml files of the steps of the transfer specify whether or not file descriptors will be added to files that are transferred. For more information about the CreateFileDescriptors parameter, refer to *Configuring File Level Tracking*, page *93*.

Note: In version 3.5 of the DCA, the File ID File Category was created on files that were transferred when the **SupportApplicationInfo** parameter in the process.xml file was set to 'True'. In version 4.1 and above, this File Category will only be created if the **CreateFileDescriptors** parameter is specified and set to 'True' in the process.xml file. The **SupportApplicationInfo** parameter will still determine whether or not application information (File Categories) will be used in the process.

File Tracking Information

The DCA uses the following file tracking properties to describe the path of a file during a transfer.

File Descriptor	Indicates …	File Category
Steps	The Vaults and Safes that the file was stored in before it was transferred to its current location.	__S1__,__S2__,…
Last Step ID	The unique ID of the last step in the file transfer path.	__LastStepID__

The **WriteTrackInfoToFiles** parameter in the process xml files of the steps of the transfer determine whether or not file tracking information will be added to files that are transferred. For more information about the WriteTrackInfoToFiles parameter, refer to *Configuring File Level Tracking*, page *93*.

Event Properties

The Event record in the Vault contains the following fields:

Field	Indicates
EventInstanceId	The unique identification of this Event record in the Vault.
SourceId	The Id of the application that created this event. The Source Id of events that are created by the DCA is 555.
EventTypeId	The type of event, as described in *Types of Events*, page *86*.
EventCreationDate	The time when the Event was created in the Vault.
EventExpirationDate	The date when the Event may be deleted from the Safe by Clear Safe History.
IssuerUserName	The name of user who created the Event.
SafeName	The name of the Safe where the Event was created.
EventData	Application data attached to the event. For more information, refer to the section below.

For a full list of fields included in the Event record, refer to Cyber-Ark's NAPI documentation.

Events can be retrieved from the Vault with Cyber-Ark API functions. For more information, refer to *Using APIs to Retrieve and Analyze Events*, page *97*.

Event Data Property

The EventData property in the Event contains a variety of information that describes the file transfer that created this Event. This information is stored in the EventData property in the following format:

<separator><name>=<value><separator><name>=<value>...<separator>

This format can be repeated as many times as required to display all the necessary information. The following character is used as the separator: "\x1E".

The following table lists the different information that appears in the EventData property:

Field name	Value
File Descriptors	
FID	The unique file id of the file that is transferred.
PID	The Process Id of the processes that transferred the file.
PIID	The Process instance Id of the first step that transferred the file.
Track Information (internal for the DCA)	
S1, S2, …	Steps that describe the track information of the file that is transferred. The format of each step is: <vault-id>;<safe-id>
LSI	The Step Id of the process that will transfer this Event to the previous Safe in the transfer.
File Transfer Basic Information	
FNM	The file name of the file that is transferred.
PNM	The name of the process that transferred the file.
PIT	The startup time of the process instance that transferred the file.
CTS	The creation timestamp of the Event.
CSI	The Step Id of the process that created the Event.
Transfer Extra Information	
SRC	The source path of the transfer.
DST	The destination path of the transfer.
ERR	An error message related to the transfer, if the transfer failed.

The **WriteExtraInfoToEvents** parameter in the process xml files of the steps of the transfer determines whether or not to add the parameters specified in the Transfer Extra Information section to Events. For more information about the WriteExtraInfoToEvents parameter, refer to *Configuring File Level Tracking*, page *93*.

Configuring File Level Tracking

The DCA uses several parameters to create Events for file transfers, as described below.

DCA Parameter File

The DCA uses the following parameters in the DCA parameter file to configure file level tracking:

Field	Indicates	Default Value	Acceptable Values
EventsThreadPoolSize	The number of process event transfers that can run simultaneously.	10	1-50
TrackInfoMaxSteps	The number of steps that the DCA supports in multiple steps transfers.	5	1-32000

For more information, refer to *Appendix A: Dca.ini*, page *104*.

Process Configuration File

The DCA uses the following parameters in the process configuration file to configure file level tracking:

Process Level Parameters

The following parameters are specified at process level:

File Information	Indicates …	Acceptable Values
ProcessID	The unique ID of the process. If this parameter is not specified, the process name will be used as the ProcessID. For more information, refer to *File Descriptors*, page *89*.	String

File Information	Indicates …	Acceptable Values
ProcessStepId	The step identification of this process in the overall transfer. For more information, refer to *File Descriptors*, page *89*.	Numeric
TransferFileEvents	Whether or not events will be transferred. Default = false For more information, refer to Consolidating Events, page 87.	True/False
TransferEventsInterval	The number of seconds that will pass between event transfers. Default = 30 For more information, refer to Consolidating Events, page 87.	Numeric

Process / Rule level parameters:

The following parameters are specified either at process or rule level:

File Information	Indicates …	Acceptable Values
CreateFileDescriptors[1]	Whether or not to create file descriptors if they do not exist in the files that are transferred. Default = false For more information, refer to *File Descriptors*, page *89*.	True/False
CreateFileEvents[1]	Whether or not the DCA will create Events. Default = false	True/False
WriteTrackInfoToFiles[1]	Whether or not to attach tracking information to files. Default = false For more information, refer to *File Tracking Information*, page *90*.	True/False

File Information	Indicates …	Acceptable Values
WriteExtraInfo ToEvents[1]	Whether or not to attach extra file information to Events. Default = true For more information, refer to *Event Data Property*, page *91*.	True/False

[1] - This parameter can be specified at either process or rule level.

Implementing File Level Tracking

The following steps instruct you how to implement File Level Tracking on the DCA.

1. Plan the file transfer:
   i. Define the source and destination of the transfer.
   ii. Define any additional steps that the file must pass through on the way.
   iii. For each step, define its source and destination.

   For more information about planning file transfers, refer to *Planning File Transfer Processes*, page *26*.
2. Configure the Vault(s) and Safe(s) that are part of the processes:
   i. Configure each Safe to support file categories.
   ii. In each Safe, or at Vault level to apply to all the Safes, define the following file categories:
      - __FileID__
      - __ProcessID__
      - __ProcessInstanceID__

- __S1__,__S2__, … - The number of steps in this file category must be the same as the value defined in the TrackInfoMaxSteps parameter in the DCA parameter file.
- __Last Step ID__

For more information, refer to *File Level Tracking* Properties, page *89*.

3. To change the default values of the following parameters, in the DCA parameter file, specify the following parameters:
   - EventsThreadPoolSize
   - TrackInfoMaxSteps

   For more information, refer to *Configuring File Level Tracking*, page *93*.
4. Configure the process configuration files that specify the process that form the complete file transfer:
   i. Specify the following parameters for each process that is part of the complete transfer:
      - ProcessID
      - ProcessStepId
      - CreateFileDescriptors
      - WriteTrackInfoToFiles

   ii. In addition, specify the following parameters for each process in the transfer that will create Events:
      - CreateFileEvents
      - WriteExtraInfoToEvents (optional)

   iii. Specify the following parameters for V2V processes to transfer Events to the previous Safe in the transfer.
      - TransferFileEvents
      - TransferEventsInterval

   For more information, refer to *Configuring File Level Tracking*, page *93*.

Using APIs to Retrieve and Analyze Events

You can use Cyber-Ark API functions in PACLI, NAPI, or XAPI to retrieve Events from the Safe where they are stored after the transfer, and see the current status of the transfer. These APIs also enable you to filter Events according to different properties, such as the type of Event or the time it was created.

You can also filter Events according to information in the EventData property, by specifying a substring contained in this property. In this way, you can filter Events according to different transfer details.

For example:

- To retrieve Events of a specific file, specify all the Events whose EventData property contains the following substring:
  `<separator>FID=<FID of relevant file><separator>`
- To retrieve Events of a specific process, specify all the Events whose EventData property contains the following substring:
  `<separator>PID=<PID of relevant process><separator>`
- To retrieve Events of a specific process instance, specify all the Events whose EventData property contains the following substring:
  `<separator>PIID=<PIID of relevant process instance><separator>`

For more information, refer to the relevant Cyber-Ark documentation.

In the following XAPI example, a user called **EventsUser** logs onto the **CompanyVault** using **password** authentication. This user opens the **TransferSafe** and opens a file called **EventsFile.txt**. Next, the user retrieves all the Events that are related to a file whose file ID is **661a404a-dec1-4d50-9bfe-4d3e4eac38f7** and print their details to the file EventsFile.txt. After all Events details have been printed to the text file, the file is closed, the user logs off, and the XAPI session is completed.

```
Private Sub MySub()
    Dim PrivateArkObj As New PrivateArk
    Dim VaultObj As Vault
    Dim SafeObj As Safe
    Dim EventsColl As Events
    Dim EventObj As PAObjectsLib.Event

    PrivateArkObj.Init

    Set VaultObj = PrivateArkObj.AddVault("CompanyVault", "1.1.1.222")
    VaultObj.Logon "EventsUser", "password"

    Set SafeObj = VaultObj.GetSafe("TransferSafe", True)

    ' open a file
    FileNum = FreeFile
    Open App.Path & "\" & "EventsFile.txt" For Output Shared As
     #FileNum

    ' get all events related to file with file-id 661a404a-dec1-4d50-
     9bfe-4d3e4eac38f7
    Set EventsColl = SafeObj.GetEvents(, , , "FID=661a404a-dec1-4d50-
     9bfe-4d3e4eac38f7")

    ' go over events collection
    For Each EventObj In EventsColl
           ' print event's details to a file
           Print #FileNum, EventObj.ID & " " & EventObj.CreationDate &
     " " & EventObj.EventTypeId & " " & EventObj.Data
    Next

    ' close the file
    Close #FileNum
```

```
    ' logoff from the vault
    VaultObj.Logoff

    PrivateArkObj.Term

    Set EventObj = Nothing
    Set EventsColl = Nothing
    Set SafeObj = Nothing
    Set VaultObj = Nothing
    Set PrivateArkObj = Nothing
End Sub
```

Running Processes from a Command Line Interface

The Distribution and Collection Agent command line interface enables you to activate processes from a command line at any of the three following activation levels:

- **Process level** – This level runs an entire process,
- **Rule level** – This level runs a specific rule in a process,
- **File level** – This level transfers a specific file.

The DCACLI Utility

```
dcacli  /mode <{SYNC, ASYNC}>
        /process <PROCESS_NAME>
        [/Rule <RULE_NAME>
        [/File <FILE_NAME>
        [/ApplicationInfo <ParmName=ParmValue;...>]]]
        [/Timeout <Seconds>]
        [/SetID <TransferID>]
        [/UserDefinedParms <ParmName=ParmValue;...>]
```

Parameter	Indicates …
/mode	Specifies whether the activation level will run synchronously or asynchronously.
SYNC	The activation level will run synchronously. A return code will be returned after the whole operation is finished. If the operation is not finished successfully, and the return code is not '0' (zero), an error message is also returned.
ASYNC	The activation level will run asynchronously. In this mode, the return code is always '1' (one). ◆ After an 'async' **process activation**, the process instance ID is returned. ◆ After an 'async' **rule activation**, the process instance ID and rule number are returned. The rule number is unique for a process instance ID. ◆ After an 'async' **file transfer**, the file ID is returned. These IDs can be monitored in the logs.
/process	The name of the process to activate.
PROCESS_NAME	The name of the process to activate. This process determines the file transfer parameters and configurations.
/Rule	The rule to activate. Specify this parameter to run a rule or a file transfer. **Note:** Do not specify this parameter if you wish to run a full process.
RULE_NAME	The name of the rule to activate.
/File	The file to transfer. **Note:** Do not specify this parameter if you wish to run a process or a rule.
FILE_NAME	The name of the source file to transfer.

Parameter	Indicates …
/ApplicationInfo	The name and value of the File Category to specify for the file to transfer to the Vault. **Note:** This is relevant for file level activation, and only if the file will be transferred into a Vault.
ParmName=ParmValue	The name and value of the File Category.
/Timeout	The operation timeout
Seconds	The number of seconds to wait for a response from the Distribution and Collection Agent. The default is an unlimited number of seconds.
/SetID	Sets the process instance ID (when running a process), the rule ID (when running a rule), or the file ID (when running a file). These IDs can be monitored in the logs.
TransferID	The ID value.
/UserDefinedParms	The user defined parameter names and values to specify for the process/rule/file. The user defined parameters can be accessed from either Perl or C++ user exits.
ParmName=ParmValue	Parameter name and value

Usage Notes

General

- A process that is scheduled to run by an external activation (ProcessScheduling=ExternalActivation) requires the **ExternalActivationLevel** parameter to specify the activation level (process/rule/file). You can only run the level that is specified in the External Activation Level parameter. If you try to run any other level, an error message will be returned.
- An automap process definition does not include a rule name because rules are created dynamically by the Distribution and Collection Agent. To run a rule level transfer of an automap process, specify the name of the corresponding source folder name in the **Rule** parameter.

Process Level

- A process that is scheduled to run at intervals (ProcessScheduling=interval) can be activated by the DCACLI utility, at process activation level.
- A process cannot be activated twice at the same time. As a result, a process that is scheduled to run automatically at a specified interval cannot be activated if the previous transfer has not finished.

Rule Level

- At rule level, two or more rules of the same process can run simultaneously. However, the same rule cannot be activated twice at the same time.

File Level

- At file level, two or more files in the same process or rule can be transferred simultaneously. However, the DCA will **not** prevent you from running the same file simultaneously.

Appendix A: Dca.ini

The Dca.ini parameter file specifies the parameters that determine how the Distribution and Collection Agent will work.

Parameter	Indicates	Required	Acceptable Values
TempFolder	The location of the temporary storage folder that is used during file transfers.	Yes	Full pathname
VaultsFolder	The full path of the folder where the Vault parameter files are stored.	Yes	Full pathname
CredFilesFolder	The full path of the folder where the user credential files are stored.	Yes	Full pathname
ProcessesFolder	The full path of the folder where the process definition files are stored.	Yes	Full pathname
EnableDebugLog	Whether or not debug records are written to the debug log and the activity log. Default = No	No	Yes/No
ProcessesAndRules ThreadPoolSize	The number of processes and rules that can run simultaneously. Default = 5	No	1-50
TransferFilesThread PoolSize	The number of files that can be transferred simultaneously. Default = 15	No	1-50
ShutdownTimeout	The maximum number of seconds to wait for process instances to finish during shutdown.	No	1-300
SocketsPoolSize	The number of sockets that will be open to the Vault. Default = 100	No	1-32000

Parameter	Indicates	Required	Acceptable Values
LogRecycleSizeMB	The maximum size in MB of the process log, after which a new log will be started. This parameter overrides the same parameter in Dca.ini for the process being defined. If this parameter is not defined in either process.xml or dca.ini, the default log size is 25MB.	No	0-1024
LogRecycleInterval Minutes	The maximum number of minutes that will pass after which a new process log will be started. This parameter overrides the same parameter in the Dca.ini configuration file for the process being defined.	No	0-60*24*366 (one year)
LogsColumnsDelimiter	The logs delimiter between columns (one character only allowed). Default = "|".	No	"<*any alphanumeric character*> "
PeriodicWorkInterval Seconds	The number of seconds between each time the DCA will check the Logs\old folder for log files that should be deleted. Log files will be deleted if the number of days specified in the KeepOldLogsDays parameter have passed since they were last used. To disable this parameter, set it to '0' (zero). Default = 0.	No	

Parameter	Indicates	Required	Acceptable Values
KeepOldLogsDays	Determines the number of days that must pass since the last time that a log file was used, so that it can be deleted. The PeriodicWorkIntervalSeconds parameter must be greater than 0 to delete log files automatically. Set this parameter to '0' (zero) to prevent log files from being deleted. Default = 0.	No	
SafesMetadataCache Retention	The number of seconds that Safes' metadata information is saved in the internal cache. Default = 300	No	Number
CommandLineThread PoolSize	The number of command line requests that can be performed simultaneously. Default = 5	No	1-50
CLIListenerPort	The port number through which the DCA will accept CLI commands. Default = 7777	No	1025-65535
EventsThreadPool Size	The number of process event transfers that can run simultaneously. Default = 10	No	1-50
TrackInfoMaxSteps	The number of steps that the DCA supports in multiple steps transfers. Default = 5	No	1-32000

Appendix B: Vault.ini

The Vault.ini file contains all the information about the Vault that the Distribution and Collection Agent will access during a transfer.

Parameter	Description	Default Value	Acceptable Values
Vault	The name of the Vault.	None	String
Address	The IP address of the Vault.	None	IP address
Port	The Vault IP Port.	1858	Number
Timeout	The number of seconds to wait for a Vault to respond to a command before a timeout message is displayed.	30	Number
ProxyType	The type of proxy through which the Vault is accessed.	None	HTTP, HTTPS, SOCKS4, SOCKS5
ProxyAddress	The proxy server IP address. This is mandatory when using a proxy server.	None	IP address
ProxyPort	The Proxy server IP Port.	8081	Number
ProxyUser	User for Proxy server if NTLM authentication is required.	None	User name
ProxyPassword	The password for Proxy server if NTLM authentication is required.	None	Password
ProxyAuthDomain	The domain for the Proxy server if NTLM authentication is required.	NT_DOMAIN_ NAME	Domain name

Parameter	Description	Default Value	Acceptable Values
ProxyCredentials	The full pathname of the credentials file that contains the Proxy user, password, and authentication domain.	None	Full pathname
BehindFirewall	Accessing the Vault via a Firewall.	No	Yes/No
UseOnlyHTTP1	Use only HTTP 1.0 protocol. Valid either with proxy settings or with BEHINDFIREWALL.	No	Yes/No
NumOfRecords PerSend	The number of file records that require an acknowledgement from the Vault server	15	Number
NumOfRecords PerChunk	The number of file records to transfer together in a single TCP/IP send/receive operation	15	Number
EnhancedSSL	Whether or not to use an enhanced SSL based connection (port 443 is required).	No	Yes/No
PreAuthSecured Session	Whether or not to enable a pre-authentication secured session.	No	Yes/No
TrustSSC	Whether or not to trust self-signed certificates in pre-authentication secured sessions.	No	Yes/No
AllowSSCFor3 PartyAuth	Whether or not self-signed certificates are allowed for 3rd party authentication (eg, RADIUS).	No	Yes/No

Appendix C: Process Definition Parameters

The process definition parameters determine the behavior of the transfer processes.

These parameters are divided into three groups:

- General process attributes
- Transfer rules
- User exits

General Process Attributes

Parameter	Description	Required	Acceptable Values
ProcessName	The name of the process. Note: This name may not contain any of the following characters: \ / < > : * “ ” |	Yes	String
ProcessID	The unique ID of the process. If this parameter is not specified, the process name will be used as the ProcessID.	No	String
ProcessStepId	The step identification of this process in the overall transfer.	No	Numeric
ProcessEnabled	Whether or not the process will run.	Yes	False/True
ProcessType	The type of process.	Yes	N2N, Automap
AutomapPattern	The pattern used to specify folders in the master source port that will be mapped during the process.	If ProcessType =Automap	String

Parameter	Description	Required	Acceptable Values
LogRecycleSizeMB	The maximum size in MB of the process log, after which a new log will be started. If this parameter is not defined in either process.xml or dca.ini, the default log size is 25MB.	Yes	0 - 1024
LogRecycleIntervalMinutes	The maximum number of minutes that will pass after which a new log will be started.		0 - 60*24*366 (one year)
NonDuplicationMethod[1]	How the file(s) will be handled after the transfer.	Yes	Delete/ Move/ AccessMark/ ArchiveBit/ None
OnFileExistsInDest[1]	How the file will be handled if a file with the same name already exists in the destination location.	Yes	KeepExisting/ KeepExistingWithWarning/ Overwrite/ OverwriteWithWarning/ Fail/ Retry/ Skip/ SkipWithWarning/ UserExit
ProcessScheduling	Whether the process will start automatically at regular intervals according to a predetermined period of time, or whether an external catalyst will activate the transfer.	Yes	Interval/ External Activation
SchedulingInterval	The interval value in seconds. This parameter is required if ProcessScheduling = Interval.	Yes	0 – 2147483647

Parameter	Description	Required	Acceptable Values
RestrictScheduling IntervalTime	Whether or not the transfer process will be restricted to a specific period of time.		True/False
SchedulingFrom Time	The time from when the transfer process will be processed. This parameter is required when the 'RestrictSchedulingIntervalTime' parameter is set to 'True'.		HH:MM,
SchedulingToTime	The time until when the transfer process will be processed. This parameter is required when the 'RestrictSchedulingIntervalTime' parameter is set to 'True'.		HH:MM,
ExternalActivation Level	Specifies the activation level of the transfer. This parameter is required if ProcessScheduling = ExternalActivation.	Yes	Process/Rule/ File
ProcessRecursive[1]	Whether or not files will be transferred from sub folders, recursively.	Yes	True/False
FileSelectionPattern[1]	The pattern for filtering the files to transfer.	Yes	'*' for any string '?' for any character
FileSelection Categories[1]	Categories for filtering the files to transfer.	No	'*' for any string '?' for any character
FileSelection Relationship BetweenCategories[1]	The logical relationship between file categories. Default = OR	No	OR/AND

Parameter	Description	Required	Acceptable Values
FileObjectsEarly Delete	Used during massive file transfers to allow internal file objects that are allocated in memory to be purged early in the process in order to reduce memory consumption. Default = True	No	True/False
RuleRetries[1]	The number of retries for transfer rules. Default = 0	No	0-2147483647
FileRetries[1]	The number of retries for file transfers. Default = 0	No	0-214748364
ResubmitJobsDelay[1]	The number of seconds to wait between retries. Default = 0	No	0-2147483647
BatchUserExits Timeout	The number of seconds to wait for batch user exits to finish. Default = 30	No	0-2147483647
LockOnRetrieve[1]	Whether or not to lock the source file when it is retrieved in a V2V/ V2FS transfer. Default = True **Note:** When this parameter is set to 'true', files that are locked by any user other than the user carrying out the transfer will not be transferred.	No	True/False

Parameter	Description	Required	Acceptable Values
LockOnFS	Whether or not to lock the source file when it is retrieved in a FS2V transfer, and the action to carry out if the lock failed.	No	None/Fail/ Skip
LockOnFSAccess Mode	The access mode to use to lock the source file in a FS2V transfer. Values: See values for dwDesiredAccess parameter of the CreateFile function in the Windows Platform SDK. Default: GENERIC_READ.	No	
LockOnFSShare Mode	The access mode to use to lock the source file in a FS2V transfer. Values: See values for dwDesiredAccess parameter of the CreateFile function in the Windows Platform SDK. Default: GENERIC_READ	No	
RetrieveResumable[1]	Whether or not to resume retrieval in a V2V/V2FS transfer. Default = True	No	True/False
ApplicationInfo Support[1]	Whether or not to read and write files application info (File Categories), including FileID's. Default = False	No	True/False
DynamicDestinations	Whether or not destination peers will accept missing details that will be completed with user exits. Default = False	No	True/False
CreateDestination Folder	Whether or not to create a destination folder. Default = True	No	True/False

Parameter	Description	Required	Acceptable Values
CreateArchiveFolder	Whether or not to create an archive folder. Default = True	No	True/False
CreateFile Descriptors[2]	Whether or not to create file descriptors if they do not exist in the files that are transferred. Default = false	No	True/False
WriteTrackInfo ToFiles[2]	Whether or not to attach tracking information to files. Default = false	No	True/False
CreateFileEvents[2]	Whether or not the DCA will create Events. Default = false	No	True/False
WriteExtraInfo ToEvents[2]	Whether or not to attach extra file information to Events. Default = true	No	True/False
TransferFile Events	Whether or not events will be transferred. Default = false	No	True/False
TransferEvents Interval	The number of seconds that will pass between event transfers. Default = 30	No	Numeric
UserDefinedParms	The user defined parameter names and values to specify for the process. This can be retrieved during user exits.	No	
UserExitsDlls	The full pathname of dlls for dll-based user exits.	No	String
UserExitDll<#>	The full path of each user exit dll.	No	Full pathname

Parameter	Description	Required	Acceptable Values
MasterPorts	In an automap process, the master source destination ports from which the dynamic rules will be created.	Yes	
MasterSourcePort	In automap process, the Master source port. Additional parameters are listed in the following table.	Yes	Refer to the following tables describing the file system or Vault port parameters.
MasterDestPort	In automap process, the Master destination port. Additional parameters are listed in the following table.	Yes	Refer to the following tables describing the file system or Vault port parameters.

[1] These parameters can be specified in the Rules section and will overrule the process configuration.

[2] These parameters can be specified at either process or rule level.

When the MasterPorts parameter is specified, the following parameters must be used to specify the MasterSourcePort and MasterDestPort.

If the master port is a file system:

Parameter	Description
Name	The logical name of the file system port.
Type	Indicates that the port is a file system
FolderName	The name of the folder where the files will be transferred to or from.
ArchivePath	If the non-duplication method is 'Move', this parameter indicates where to move the source files after they have been transferred. **Note:** This parameter is applicable to the SourcePort definitions.

If the master port is a Vault:

Parameter	Description
Name	The logical name of the Vault port.
Type	Indicates that the port is a Vault.
VaultName	The name of the source or destination Vault from/to where the files will be transferred.
SafeName	The name of the Safe where the files are stored.
FolderName	The name of the folder where the files will be transferred to or from.
UserName	The name of the Vault user who will authenticate to the Vault and access the files to transfer.
ArchivePath	If the non-duplication method is 'Move', this parameter indicates where to move the source files after they have been transferred. **Note:** This parameter is applicable to the SourcePort definitions.

Transfer Rules

Parameter	Description	Required
RuleName	The name of the rule that will be defined.	No
UserDefinedParms	The user defined parameter names and values to specify for the rule. This can be retrieved during user exits.	No
SourcePort	The source location from where files will be transferred.	Yes
DestPort	The destination location to where files will be transferred.	Yes

If the SourcePort or DestPort parameters are specified, the following parameters specify their specific locations:

If the port is a file system:

Parameter	Description
Type	Indicates that the port is a file system
Folder	The name of the folder where the files will be transferred to or from.
ArchivePath	If the non-duplication method is 'Move', this parameter indicates where to move the source files after they have been transferred. **Note:** This parameter is applicable to the SourcePort definitions.

If the port is a Vault:

Parameter	Description
Type	Indicates that the port is a Vault.
VaultName	The name of the source or destination Vault from/to where the files will be transferred.
SafeName	The name of the Safe where the files are stored.
Folder	The name of the folder where the files will be transferred to or from.

Parameter	Description
UserName	The name of the Vault user who will authenticate to the Vault and access the files to transfer.
ArchivePath	If the non-duplication method is 'Move', this parameter indicates where to move the source files after they have been transferred. **Note:** This parameter is applicable to the SourcePort definitions.

User Exits

The following table lists the available user exits:

Parameter	Indicates …
PreProcess	Before the process instance starts running, process attributes can be changed.
PostProcess	After the process instance has finished running.
PreRule	Before the transfer rule starts running.
PostRule	After the transfer rule is finished.
PostFindFiles	After the transfer rule locates the files to transfer, groups of files can be managed together. For example, multiple files can be stored in one zip file.
OnFileInit	When the transfer file is initialized.
PreFile	Before the file transfer process starts running.
PostRetrieveFile	In a Vault-to-Vault transfer after the file is retrieved from the source Vault, a user exit can be activated before it is stored in the destination Vault.
OnFileExistsInDestination	After the file has been transferred to the destination location.
PostFile	After the file transfer process has been completed, a user exit can be activated to change the properties of the destination file.
OnError	When an error occurs.

For each user exit, the following parameters must be specified:

Parameter	Description	Required	Acceptable Values
UserExitType	The type of user exit.	Yes	Batch/ Perl/ Dll
UserExitFileName	The file name of the user exit. This is relevant for Batch and Perl user exits.	Yes	Valid path of a file
UserExitFuncName	If UserExitType=Perl, specify the Perl function name. If this parameter is not specified, a function with the same name as the user exit will be executed.	No	String
UserExitUsesVaultSession	If UserExitType=batch, specify whether or not the batch user exit will use a session to access the source or destination Vaults.	Yes	True / False
UserExitDllReference	If UserExitType=dll, specify the reference to the full pathname of the dll.	Yes	String

When you use C++ user exits, specify the following parameters in the general process attributes of the process file.

Parameter	Description
UserExitsDlls	The full pathnames of dlls for C++ user exits.
UserExitDll<#>	The full pathname of each user exit dll.

Appendix D: Creating a User Credential File

The Distribution and Collection Agent carries out processes by logging onto the Vault with a user credential file that contains the user's Vault username and encrypted logon information.

This user credential file can be created for password, RADIUS, Token, or PKI authentication with a utility that is run from a command line prompt. This utility, CreateCredFile, is located in the CyberArk\Utilities installation folder. It can also create a user credential file for authentication through a Proxy server.

Before creating the user credential file, make sure that you are familiar with the user's authentication details in the Vault.

The CreateCredFile utility uses the following syntax:

```
CreateCredFile <FileName> <command> [command parameters]
```

Parameter	Specifies
Filename	The name of the user credential file to create or update, specifically **user.cred**.
Password	Indicates that the credential file will be created with password authentication details.
/Username	Sets the username in the credential file. This parameter is required. If you do not specify it in the command, you will be prompted for it.
/Password	The password that will be encrypted in the credential file. This parameter is required. If you do not specify it in the command, you will be prompted for it.
/Radius	Creates a user name-password credential file for use with RADIUS server.

Parameter	Specifies
Token	Creates a user credential file with a key stored on a token.
/Username	Sets the username in the credential file. This parameter is required. If you do not specify it in the command, you will be prompted for it.
/Password	The password that will be encrypted in the credential file. This parameter is required. If you do not specify it in the command, you will be prompted for it.
/DLLpath	Specifies the DLL file path used by the token device. This parameter is required. If you do not specify it in the command, you will be prompted for it.
/PIN	Specifies the PIN code required by the token device. This parameter is required. If you do not specify it in the command, you will be prompted for it.
/Radius	Creates a credential file for use with RADIUS server.
/InitToken	Specifies that the token will be initialized.
PKI	Creates a credential file based on a PKI certificate.
/CertIssuer	Personal certificate issuer.
/CertSerial	Personal certificate serial number.
/PIN	Specifies the PIN code required to access the certificate. This parameter is required if the certificate is stored on a Token.
PROXY	Creates a credential file based on PROXY authentication.
/ProxyUser	The name of the Proxy user. This parameter is required. If you do not specify it in the command, you will be prompted for it.
/ProxyPassword	The password that will be decrypted in the credential file. This parameter is required. If you do not specify it in the command, you will be prompted for it.
/ProxyAuth Domain	The domain name of the Proxy user.
/?	Lists the available options.

The following instructions explain how to create a user credential file. The examples used in these instructions run the utility from the DCA\Utilities folder, and create a credential file called ‘user.cred’.

Note: The text typed by the user appears in bold.

Creating the User Credential File for Password Authentication

1. At the command line prompt, run the **CreateCredFile.exe** utility. You must specify the username and password to the Vault. You can also specify whether or not Radius authentication will be used.

```
C:\Program Files\CyberArk\DCA\Utilities> createcredfile.exe DCAuser.cred
password /username DCAuser /password abcdef /radius
```

   The above example shows that this credential file will be called ‘DCAuser.cred’, and will contain an encrypted password for the Vault user called ‘DCAuser’. The file can be used to log onto the file with Radius authentication.

   If you do not specify the command parameters, username, password, and radius, you are prompted for them now. An example of this appears in the following example:

```
Vault Username [mandatory] ==> DCAuser
Vault Password (will be encrypted in credential file) ==> *******
Radius server will be used for authentication (yes/no) [y] ==> yes
```

   The user’s credential file will now be created and saved in the current folder.

```
Command ended successfully
```

Creating the User Credential File using a Token

The Vault supports logon with a password that has been encrypted by a key on a USB token or a Smartcard. This password is stored in the user's credential file, and is decrypted by the external token for logon.

Any PKCS#11 token can be used for this type of authentication, as long as it meets all of the following criteria:

- The token must be a hardware token.
- The token is accessible through the PKCS#11 interface.
- Access to the token is only possible after supplying a PIN.
- The token supports RSA with 1024 or 2048 bit key length.
- The token must be able to perform encryption and key generation in hardware.

These instructions are for creating a user credential file with a new external token.

1. Attach the token to the computer.
   - If you are using a USB token, place the token in the USB port.
   - If you are using a Smartcard, place the card in the Smartcard reader.

2. At the command line prompt, run the **CreateCredFile.exe** utility. You must specify the username and password to the Vault, the full path of the PKCS#11 dll file that will encrypt the password, and the PIN that is required by the token device. You can also specify

```
C:\Program Files\PrivateArk\Server> CreateCredFile.exe DCAuser.cred
token /username DCAuser /password asdf /dllpath
i:\windows\system32\etpkcs11.dll /pin 12341234
```

   The above example shows that this credential file will be called 'DCAuser.cred', and will be created with a key that is stored on a token. 'DCAuser' is the user who will be specified in the credential file, together with his password, asdf. The dll path used by the token device is specified, as well as the PIN that is required to access the token device.

   If you have not specified the username, password, dll path and password, you are prompted for it now.

```
Vault Username [mandatory] ==> DCAuser
Vault Password (will be encrypted in credential file) ==> ****
Path of Token dll [mandatory] ==> i:\windows\system32\etpkcs11.dll
Pin code required by the Token device ==> ********
Radius server will be used for authentication (yes/no) [optional] ==> no
Initialize the Token (yes/no) [optional] ==> no
```

3. To initialize the token, type **yes**,

   or,

   If the token has already been initialized with the CreateCredFile utility, type **no**.

   The user credential file is now created and saved in the current folder.

```
Command ended successfully
```

Creating the User Credential File for PKI Authentication

The user can create a user credential file for logon with a PKI certificate. Before creating the credential file, the authentication certificate must be imported into the Microsoft Windows certificate store.

When using the Distribution and Collection Agent with PKI Authentication, the DCA Service must use the windows account whose certificate store contains the certificate that is used for authentication.

For more details, refer to *Importing a Certificate for Authentication*, page *126*.

Note: A PIN to access a PKI certificate can only be used in a Windows 2000 environment or higher.

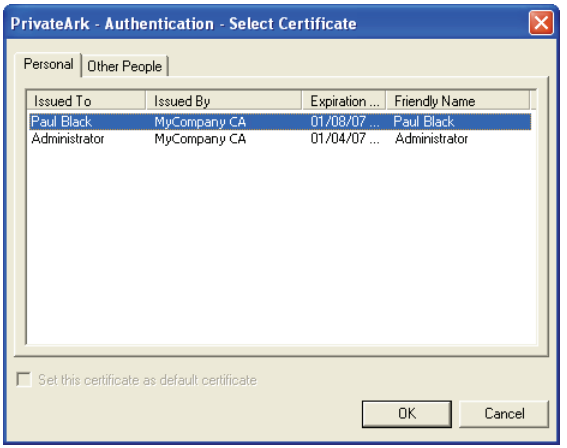
- At the command line prompt, run the **CreateCredFile.exe** utility.

```
C:\Program Files\PrivateArk\Server> createcredfile.exe Simon.cred pki
createcredfile.exe Simon.cred pki /certissuer "CN=MyCompany_CA"
/certserial "1963f68d00000000017c" /pin 12341234
```

The above example shows that this credential file will be called 'Simon.cred', and will be created based on a PKI certificate. The certificate issuer for this credential file is MyCompany_CA and the certificate detail serial number is '1963f68d00000000017c'. The PIN required to access this certificate is '12341234'.

If you do not specify the certificate issuer and serial number, the Select Certificate window appears to enable you to select the PKI certificate that will give the user access to the Vault.

Note: If a PIN is required to access the certificate, you must enter the PIN in the command line.



- Select the PKI certificate to use, then click **OK**; the user's credential file will now be created and saved in the current folder.

  The following message appears to confirm that the authentication file has been created successfully.

```
Command ended successfully
```

For details about configuring the Vault and the user to work with PKI authentication, refer to the Cyber-Ark Vault User's Guide.

Importing a Certificate for Authentication

Authentication certificates can be used to authenticate to the Vault if the certificate has been imported into the Microsoft Windows certificate store.

The certificate store is divided into several locations to limit accessibility (for security reasons). The most common location for certificates is the "Current User" location. When importing certificates into Microsoft Windows, this is the default location into which the certificates are imported. The certificates in the "Current User" location are only accessible to the user that is currently logged on. One user will not be able to access certificates in another user's "Current User" location.

Creating the User Credential File for Proxy Authentication

The Proxy user and password can be stored encrypted in a credentials file instead of being specified in the Vault parameter file.

1. At the command line prompt, run the **CreateCredFile.exe** utility.

```
C:\Program Files\PrivateArk\Server> CreateCredFile.exe PUser.cred proxy
/proxyuser PUser /proxypassword abcd /ProxyAuthDomain MyCompany.com
```

The above example will create a file called 'PUser.cred' and will enable the proxy user to log onto the Vault with proxy authentication. The credentials file will contain an encrypted proxy password for the proxy user called PUser on a proxy authentication domain called 'MyCompany.com'.

If you do not specify the name and password of the proxy user, you will be prompted for them. An example of this appears in the following example:

```
Proxy Username [mandatory] ==> PUser
Proxy Password (will be encrypted in credential file) ==> ****
Domain name of ProxyUser [optional] ==> MyCompany.com
```

The user's credential file will now be created and saved in the current folder.

```
Command ended successfully
```

Appendix E: Glossary

DCA

A service that facilitates file transfers between the Cyber-Ark Vault and file servers and between two Vaults.

Process

A set of definitions that determine the file transfer operation.

Rule

A subset of a process that defines a specific source path and a specific destination path. A process may contain more than one rule.

File Transfer

A transfer operation of a single file within a rule context.

Process Instance

A process that is currently carrying out file transfers based on preset rules.

Appendix F: Messages

Service Messages

Code: CFTSR001I
Message: CAFT Service is starting...

Code: CFTSR002I
Message: CAFT Service is shutting down.

Code: CFTSR003I
Message: CAFT Service was stopped.

Code: CFTSR010I
Message: CAFT Service was started successfully.

Code: CFTSR023E
Message: Failed to load Vault <Vault>.

Code: CFTSR024W
Message: Invalid file <file> was found under Vaults folder <folder>.

Code: CFTSR065W
Message: Warnings occurred when trying to load all Vaults.

Code: CFTSR066E
Message: Not all processes were loaded.

Code: CFTSR067E
Message: Could not initialize XML lib.

Code: CFTSR068E
Message: Configuration file <file> could not be loaded.

Code: CFTSR069E
Message: Configuration file <file> is not valid.

Code: CFTSR070E
Message: Failed to initialize thread pool.

Code: CFTSR071E
Message: Failed to create log file <file>.

Code: CFTSR073E
Message: Failed to find Vault ini file of Vault <vaultname> (file name should be <*vaultname*>.ini or vault.ini).

General Messages

Code: CFTGE001E
Message: Failed to read parameter <parameter> for file <file>. The parameter does not exist or is invalid.

Code: CFTGE002E
Message: Failed to load xml parser.

Code: CFTGE003E
Message: Failed to read xml file <file>.

Code: CFTGE004E
Message: Parameter <parameter> is invalid: folder <folder> does not exist.

Code: CFTGE005E
Message: Parameter <parameter> is invalid: invalid folder path <path>.

Code: CFTGE006E
Message: Parameter <parameter> is invalid: could not convert <value> to a valid number.

Code: CFTGE007E
Message: Parameter <parameter> is invalid: value must be between <number> and <number>.

Code: CFTGE008E
Message: Parameter <parameter> is invalid: value is empty.

Code: CFTGE009E
Message: Parameter <parameter> is invalid: value cannot include the characters.

Code: CFTGE010E
Message: Parameter <parameter> is invalid: file <file> does not exist.

Code: CFTGE011E
Message: Parameter <parameter> in file <file> is illegal.

Session Messages

Code: CFTSE001E
Message: Create session failed. Vault=<vault>, User=<user>.

Code: CFTSE002E
Message: Delete session failed. Vault=<vault>, User=<user>.

Code: CFTSE003E
Message: Deletion of master session failed. Vault=<vault>, User=<user>.

Code: CFTSE004E
Message: Session logon failed. Vault=<vault>, User=<user>.

Code: CFTSE005E
Message: Session logoff failed. Vault=<vault>, User=<user>.

Code: CFTSE006E
Message: Failed to register callbacks handler for session. Vault=<vault>, User=<user>.

Process Factory Messages

Code: CFTPF001E

Message: ProcessFactory unknown process type <process type value>.

Process Manager Messages

Code: CFTPM001I

Message: Process manager is loading all processes <file pattern>.

Code: CFTPM002I

Message: Process manager has loaded all processes.

Code: CFTPM011I

Message: Process manager is loading process file <file>

Code: CFTPM012I

Message: Process <process name> was loaded successfully.

Code: CFTPM013I

Message: Process file <file> is disabled.

Code: CFTPM021E

Message: Failed to add process <process name> to processes map. Check that each process has a unique name. Process will not run.

Code: CFTPM022E
Message: Failed to load process configuration file <file>. Process will not run.

Code: CFTPM023E
Message: Process <process name> is invalid. Process will not run.

Code: CFTPM024E
Message: Process <process name> was not found. Process will not run.

Code: CFTPM025E
Message: Process <process name> is not set for external activation. File transfer requests for this process will not run.

Code: CFTPM026E
Message: Process <process name> is already running.

Code: CFTPM027E
Message: Rule <rule name> was not found in the process configuration of process <process name>.

Code: CFTPM028E
Message: Process <process name> won't run because shutdown was called.

Code: CFTPM029E
Message: File <file> of process <process name> won't run because shutdown was called.

Code: CFTPM030E
Message: Specify Rule Name.

Code: CFTPM031E

Message: Rule <rule name> of process <process name> won't run because shutdown was called.

Code: CFTPM032E

Message: Process instances of process <process name> did not finish normally.

Code: CFTPM033E

Message: Cannot run an entire process based on the configuration file of process <process name>. Set ExternalActivationLevel=Process.

Code: CFTPM034E

Message: Cannot run a single Rule based on the configuration file of process <process name>. Set ExternalActivationLevel=Rule.

Code: CFTPM035E

Message: Cannot run a single File based on the configuration file of process <process name>. Set ExternalActivationLevel=File.

Code: CFTPM036E

Message: Failed to reload process based on the configuration file of process <process name>.

Code: CFTPM037E

Message: Cannot run process <process name>, because it is waiting to be reloaded.

Code: CFTPM038E

Message: Application data cannot be added to a destination file on a file system.

Process Messages

Code: CFTPR001I

Message: New <process type> process <process name> has started. InstanceID <instance ID>.

Code: CFTPR002I

Message: Process <process name> is trying again (tries left: <number>).

Code: CFTPR003I

Message: Process <process name> is starting its post-operations.

Code: CFTPR004W

Message: Process <process name> finished with <number> transfer rules warnings.

Code: CFTPR005E

Message: Process <process name> error: <number> rules failed.

Code: CFTPR006E

Message: Process <process name> was finished with errors. No transfer rules were sent at all.

Code: CFTPR007E

Message: Process <process name> was finished with errors. ([Rules failed: <number>] [Rules warned: <number>] [Rules succeeded: <number>])

Code: CFTPR008E

Message: Process <process name> has failed. No transfer rules were sent at all.

Code: CFTPR009E

Message: Process <process name> has failed. ([Rules failed: <number>] [Rules warned: <number>] [Rules succeeded: <number>])

Code: CFTPR010I

Message: Process <process name> completed successfully ([Rules succeeded: <number>] [Rules warned: <number>])

Code: CFTPR011W

Message: Process <process name> completed with warnings ([Rules succeeded: <number>] [Rules warned: <number>])

Code: CFTPR012W

Message: Process <process name> completed with warnings. No transfer rules were sent at all.

Code: CFTPR023E

Message: Process Instance of process <process name> failed to add rule job to thread pool.

Code: CFTPR024E

Message: Process Instance of process <process name> failed to resubmit itself.

Code: CFTPR045E

Message: Failed to register log file of process <process name>.

Code: CFTPR051E

Message: Process <process name> Vault data port <port> does not support non-duplication method 'Archive-Bit' (which is supported only on file-system data port).

Code: CFTPR052E

Message: 'Archive-Bit' non-duplication method is supported only in Windows.

Code: CFTPR053E

Message: Process <process name> FileSystem data port <port> does not support non-duplication method 'Access-Mark' (which is supported only in Vault data port).

Code: CFTPR054E

Message: Process <process name> data port <port> is missing Archive Path ('Move' non-duplication method has specified).

Code: CFTPR055E

Message: User exits dll path <path> in process <process name> is invalid

Code: CFTPR056E

Message: Failed loading user exits dll <dll name> in process <process name> (system error <error>).

Code: CFTPR057E

Message: Failed to insert user exits dll <dll name> in process <process name> to user exits dlls map

Code: CFTPR058E

Message: Dll-based user exit <user exit> has an invalid dll reference

Code: CFTPR201E

Message: Process <process name> has invalid data ports. Process will not run.

Code: CFTPR202E

Message: Process <process> contains an invalid rule (<source peer>-> <destination peer>): transfer from <peer type> data port to <peer type> data port is not supported in <process type> process type.

Code: CFTPR203E

Message: Process <process name> failed to add user exit to map.

Code: CFTPR231E

Message: Process <process name> destination data port <port> cannot be below the source data port <port> (process is recursive).

Code: CFTPR232E

Message: Process <process name> failed to automap source folders to destination folders.

Code: CFTPR233E

Message: Rule <rule name> already running in process <process name>.

Code: CFTPR234I

Message: Process <process name> was skipped

Code: CFTPR235E

Message: Process <process name> destination data port <port> cannot be equal to the source data port <port> (Cannot copy a folder to itself).

Code: CFTPR236E

Message: Failed to GetProcAddress of UserExitsDllEntryFunc in user exits dll <dll name> in process <process name> (system error <error>).

Code: CFTPR237E

Message: Entry proc of user exits dll <dll name> in process <process> returned null (expecting a pointer to IFTUserExitsHandler).

Code: CFTPR238E

Message: Exception occurred while trying to load or GetProcAdress for user exit dll <dll name> in process <process name>.

Running Object Messages

Code: CFTRO001E

Message: RunningObject: Add job failed.

Code: CFTRO009E

Message: RunningObject operation failed: <number> children jobs have failed.

Code: CFTRO010I

Message: RunningObject from type <running object type> exited with failure because of shutdown.

Code: CFTRO011W

Message: RunningObject operation warning: <number> children jobs have been warned.

Transfer Rule Messages

Code: CFTXR001I

Message: New <transfer rule type> transfer rule started: <from path> ===> <to path>

Code: CFTXR002I

Message: Transfer rule is trying again (tries left: <number>).

Code: CFTXR003I

Message: Transfer rule is starting its post-operations.

Code: CFTXR004W

Message: Transfer rule finished with <number> files warnings.

Code: CFTXR005E

Message: Transfer rule error: <number> files transferred failed.

Code: CFTXR006E

Message: Transfer rule finished with errors. No files were transferred.

Code: CFTXR007E

Message: Transfer rule finished with errors. ([Files failed: <number>] [Files warned: <number>] [Files succeeded: <number>])

Code: CFTXR008E

Message: Transfer rule has failed. No files were transferred.

Code: CFTXR009E

Message: Transfer rule has failed. ([Files failed: <number>] [Files warned: <number>] [Files succeeded: <number>])

Code: CFTXR010I
Message: Transfer rule completed successfully. ([Files succeeded: <number>] [Files warned: <number>])

Code: CFTXR011W
Message: Transfer rule completed with warnings. ([Files succeeded: <number>] [Files warned: <number>])

Code: CFTXR012W
Message: Transfer rule completed with warnings. No files were transferred.

Code: CFTXR021I
Message: Creating session with Vault <Vault> ...

Code: CFTXR022I
Message: Session created successfully.

Code: CFTXR031E
Message: Rule failed to add job.

Code: CFTXR032E
Message: Rule failed to resubmit itself.

Code: CFTXR051I
Message: Rule found no files to transfer.

Code: CFTXR052E
Message: Cannot add transfer file to rule after a rule has already started to transfer files

Code: CFTXR053I
Message: Transfer rule was skipped

Code: CFTXR612E
Message: Rule failed to find files.

Code: CFTXR617E
Message: Destination Safe <Safe > is missing file category __FileID__ (which is necessary for Application Info).

Code: CFTXR619E
Message: User defined parameter <parameter> does not exist.

Transfer File Messages

Code: CFTXF001I
Message: New <transfer file type> transfer file started: FileID=<file ID> <from path> ===> <to path> , (tid=<thread ID>)

Code: CFTXF002I
Message: Transfer file is trying again (tries left: <number>) (tid=<thread ID>).

Code: CFTXF003I
Message: Transfer file is starting its post-operations.

Code: CFTXF004I
Message: Transfer file updated destination: <destination>.

Code: CFTXF007E
Message: Transfer file error.

Code: CFTXF009E
Message: Transfer file has failed.

Code: CFTXF010I
Message: Transfer file completed successfully <action>

Code: CFTXF011W
Message: Transfer file completed with warning.

Code: CFTXF024E
Message: Failed to connect to Vault <Vault> with user <user>.

Code: CFTXF030E
Message: File exists in destination. (OnFileExistsInDest=Retry)

Code: CFTXF031I
Message: File will not be transferred, because file exists in destination. (OnFileExistsInDest=KeepExisting)

Code: CFTXF032W
Message: File will not be transferred, because file exists in destination. (OnFileExistsInDest=KeepExistingWithWarning)

Code: CFTXF033I
Message: File will be overwritten, because file exists in destination. (OnFileExistsInDest=Overwrite)

Code: CFTXF034W
Message: File will be overwritten, because file exists in destination. (OnFileExistsInDest=OverwriteWithWarning)

Code: CFTXF035E
Message: Transfer failed, because file exists in destination. (OnFileExistsInDest=Fail)

Code: CFTXF036I
Message: File will not be transferred, because file exists in destination. (OnFileExistsInDest=Skip)

Code: CFTXF037W
Message: File will not be transferred, because file exists in destination. (OnFileExistsInDest=SkipWithWarning)

Code: CFTXF301E
Message: Unsupported non-duplication method <value>

Code: CFTXF302E
Message: Source file has invalid file path <path>.

Code: CFTXF303E
Message: Destination file has invalid file path <path>.

Code: CFTXF304E
Message: Cannot set decrypt / don't decrypt for file <file> after it has been retrieved.

Code: CFTXF305E
Message: Cannot set file id after operations phase is finished.

Code: CFTXF321I
Message: Non-Duplication mechanism: Deleting source file <source file>.

Code: CFTXF322I
Message: Non-Duplication mechanism: Moving source file <source file> to archive folder <folder>.

Code: CFTXF327E
Message: Non-Duplication mechanism failed to move the source file to archive folder <folder>.

Code: CFTXF328E
Message: Non-Duplication mechanism failed to reset the source Vault file access mark.

Code: CFTXF329E
Message: Non-Duplication mechanism failed to delete the source file.

Code: CFTXF610I
Message: File is set with a new FileID <file ID>.

Code: CFTXF611I
Message: FileID is copied from file source data port FileID <file ID>.

Code: CFTXF613E
Message: Failed to write application info <application info> to the Vault.

Code: CFTXF614E
Message: Can't use application info, because ApplicationInfoSupport is off.

Code: CFTXF615E
Message: Can't use application info, because the destination data port is not a Vault.

Code: CFTXF616I
Message: Transfer file was skipped.

Code: CFTXF617E
Message: Could not find source file <file> in the source folder <folder>.

Code: CFTXF618W
Message: Failed to remove file <file>.

CFTFile Messages

Code: CFTFL025E
Message: Failed to create FS folder <folder> to retrieve file from source Vault data port.

Code: CFTFL026E
Message: Failed to retrieve file from source Vault data port.

Code: CFTFL027E
Message: Failed to store file in destination Vault data port.

Code: CFTFL028E
Message: Failed to reset file <file> access mark in Vault data port.

Code: CFTFL029E
Message: Failed to unlock file <file> in Vault data port.

Code: CFTFL121E
Message: Folder name is illegal <folder>.

Code: CFTFL123E
Message: Failed to create folder <folder> in destination Vault data port.

Code: CFTFL611E
Message: Application info <application info> does not exist.

Code: CFTFL612W
Message: Application info <application info> is already marked for deletion.

Code: CFTFL613E
Message: FileID cannot be set or deleted. It has already been determined on the file transfer initialization.

Data-Peer Messages

Code: CFTDP001E
Message: Data port type <type> is not supported.

Code: CFTDP002E
Message: Process <process name> data port <port>: Vault <Vault> was not loaded.

Code: CFTDP003E
Message: Process <process name> data port <port>: invalid folder path <path>.

Code: CFTDP011E
Message: Files list path <path> failed. Rc = <code>

Code: CFTDP012E
Message: Find files <folder> failed. Rc = <code>

Code: CFTDP021E
Message: Data port <data port name> is not accessible.

Code: CFTDP022E
Message: Data port <port> folder <folder> does not exist.

Code: CFTDP031E
Message: Data port <port> archive folder <folder> cannot be below the source data port folder <folder> (process is recursive).

Code: CFTDP032E
Message: Data port <port> archive folder <folder> cannot be equal to source data port folder <folder>.

Code: CFTDP033E
Message: Failed to create folder <folder> of data port <data port>.

User Exit Messages

Code: CFTUE001E
Message: Failed to allocate perl interpreter.

Code: CFTUE002E
Message: Failed to parse perl script <script>.

Code: CFTUE003E
Message: Perl script <script> execution failed in function <function>: <message>.

Code: CFTUE00 4E
Message: Function <function> in perl script <script> should return one value but returned <code>.

Code: CFTUE005E
Message: Function <function> in perl script <script> finished with errors. Rc=<code>.

Code: CFTUE006I
Message: Function <function> in perl script <script> finished successfully.

Code: CFTUE007E
Message: Function <function> in perl script <script> caused an exception.

Code: CFTUE008W
Message: Function <function> in perl script <script> finished with warnings. Rc=<code>.

Code: CFTUE009E
Message: Batch script <script> has timed out. Rc=<code>.

Code: CFTUE010E
Message: Batch script <script> finished with errors. Rc=<code>.

Code: CFTUE011I
Message: Batch script <script> finished successfully.

Code: CFTUE012W
Message: Batch script <script> finished with warnings. Rc=<code>.

Code: CFTUE013E
Message: Batch script <script> - process could not be created. Rc=<code>.

Code: CFTUE014E
Message: Dll user exit <user exit> in dll <dll name> finished with errors. Rc=<code>.

Code: CFTUE015I
Message: Dll user exit <user exit> in dll <dll name> finished successfully.

Code: CFTUE016E
Message: Dll user exit <user exit> in dll <dll name> caused an exception.

Code: CFTUE017W
Message: Dll user exit <user exit> in dll <dll name> finished with warnings. Rc=<code>.

Code: CFTUE018W
Message: User exit <user exit> is not supported in dll-based user exits. (used in <dll>)

Code: CFTUE519I
Message: Could not find PerlUE.dll, or Perl was not installed. Rc = <code>

Code: CFTUE020E
Message: PerlUE.dll is invalid, Rc = <code>

Code: CFTUE021E
Message: User exit <user exit> is not supported in perl-based user exits. (used in <script>)

Code: CFTUE022E
Message: Function <function> in perl script <script> caused an unknown error.

Code: CFTUE023E
Message: User exit <user exit> is not supported in batch-based user exits. (used in <script>)

Code: CFTUE024E
Message: Batch User exit <user exit> cannot use Vaults sessions. (used in <script>)

User Exit Factory Messages

Code: CFTUF001E

Message: User exit type <user exit> is not supported. User exit cannot be created.

DCA Main Messages

Code: DCAMA001E

Message: Failed to create default logs folder <folder>. Service won't start.

DCA Service Messages

Code: DCASV001E

Message: DCA System Error occurred.

Code: DCASV002E

Message: Failed to initialize command line thread pool.

Code: DCASV003E

Message: Failed to add listener job to command line thread pool.

CLI Listener Messages

Code: DCALR001E
Message: Command Line Listener failed to receive command data.

Code: DCALR002E
Message: Command Line Listener received bad parameters: <message>.

Code: DCALR003I
Message: DCA Command Line Listener is shutting down.

Code: DCALR004E
Message: Command Line Listener failed to bind on requested port <port>.

Code: DCALR005E
Message: Command Line Listener - listen() failed. errno = <code>.

Code: DCALR006E
Message: Command Line Listener failed to create socket on requested port <port number>. errno = <code>.