flexera

FlexNet Manager Suite 2023 R2

Gathering FlexNet Inventory

Legal Information

Document Name: Gathering FlexNet Inventory version 2023 R2 (for on-premises implementation)

Part Number: FMS-21.0.0-FA01

Product Release Date: November 15, 2023

Copyright Notice

Copyright © 2023 Flexera.

This publication contains proprietary and confidential technology, information and creative works owned by Flexera and its licensors, if any. Any use, copying, publication, distribution, display, modification, or transmission of such publication in whole or in part in any form or by any means without the prior express written permission of Flexera is strictly prohibited. Except where expressly provided by Flexera in writing, possession of this publication shall not be construed to confer any license or rights under any Flexera intellectual property rights, whether by estoppel, implication, or otherwise.

All copies of the technology and related information, if allowed by Flexera, must display this notice of copyright and ownership in full.

FlexNet Manager Suite incorporates software developed by others and redistributed according to license agreements. Copyright notices and licenses for this externally-developed software are provided in the link below.

Intellectual Property

For a list of trademarks and patents that are owned by Flexera, see http://www.flexera.com/intellectual-property. All other brand and product names mentioned in Flexera products, product documentation, and marketing materials are the trademarks and registered trademarks of their respective owners.

Restricted Rights Legend

The Software is commercial computer software. If the user or licensee of the Software is an agency, department, or other entity of the United States Government, the use, duplication, reproduction, release, modification, disclosure, or transfer of the Software, or any related documentation of any kind, including technical data and manuals, is restricted by a license agreement or by the terms of this Agreement in accordance with Federal Acquisition Regulation 12.212 for civilian purposes and Defense Federal Acquisition Regulation Supplement 227.7202 for military purposes. The Software was developed fully at private expense. All other use is prohibited.

Gathering FlexNet Inventory

Optimizing your software licenses requires that you can balance your purchased license entitlements against consumption of those entitlements. Consumption is calculated by examining hardware and software *inventory* — details about your computing devices as well as the applications installed (or used) on them.

Inventory can be brought into FlexNet Manager Suite in four main ways:

- Inventory collected by third-party tools can be imported through adapters that normalize the data for use in FlexNet Manager Suite. There are several adapters supplied as standard (for example, see *FlexNet Manager Suite Inventory Adapters and Connectors Reference*), and it is also possible to build custom adapters to import data from other third-party tools (see *The Inventory Adapter Studio* in the same document).
- Some systems provide APIs that FlexNet Manager Suite can interrogate for inventory information.
- FlexNet Manager Suite also has a "native" ability to collect general-purpose inventory information, collectively called "FlexNet inventory" to distinguish it from inventory gathered through the other sources.
- In the specialized Kubernetes environment, FlexNet Manager Suite supports two independent, specialized agent for collecting inventory from Kubernetes environments. There is also a special tool to use "static analysis" (that is, to complete the process without interfering with any product containers) collecting software inventory from container images.

This document covers the various agents that are available for FlexNet Manager Suite, known as the FlexNet inventory agent (covered in Part 1), and the Flexera Kubernetes inventory agent and lightweight Kubernetes agent (covered in Part 2). The specialized script called `imgtrack`, built to run against the images that can instantiate containers and analyse the software that is available in the resulting containers, is covered in part 3.

Contents

Part I. The FlexNet Inventory Agent 14

1. Understanding What, Where, How, and Why 16

What Can Be Used for FlexNet Inventory Collection 17

Agent Architecture 21

Deployment Overview: Where to, and How 24

Typical Scenarios and Use Cases 28

Support for IPv6 Networks 29

2. Adopted: Details 33

Adopted: Normal Operation 33

Adopted: Services on UNIX 37

Adopted: System Requirements 38

Adopted: Accounts and Privileges 43

Adopted: Implementation 45

Adopted: Specifying an Installed Agent Upgrade 50

Adopted: Troubleshooting Inventory 52

3. Agent Third-Party Deployment: Details 56

Agent Third-Party Deployment: Normal Operation 56

Agent Third-Party Deployment: Services on UNIX 60

Agent Third-Party Deployment: System Requirements 61

Agent Third-Party Deployment: Accounts and Privileges 66

Agent Third-Party Deployment: Implementation 67

Agent Third-Party Deployment: Collecting the Software 68

Agent Third-Party Deployment: Configuring Installation on Microsoft Windows 70

Agent Third-Party Deployment: Configuring Installations on UNIX-like Platforms 76

Agent Third-Party Deployment: Protecting Your Customizations 102

Agent Third-Party Deployment: Checking the Installed Version 103

Agent Third-Party Deployment: Troubleshooting Inventory 105

4. Zero-Footprint: Details 109

Zero-Footprint: Normal Operation 109

Zero-Footprint: Non-Root Accounts 114

Zero-Footprint: System Requirements 115

Zero-Footprint: Accounts and Privileges **121**
Zero-Footprint: Implementation **122**
Zero-Footprint: Troubleshooting Inventory **125**
5. FlexNet Inventory Scanner: Details **127**
FlexNet Inventory Scanner: Normal Operation **127**
FlexNet Inventory Scanner: Operation on Windows 127
FlexNet Inventory Scanner: Operation on UNIX-Like Platforms 129
FlexNet Inventory Scanner: System Requirements **130**
FlexNet Inventory Scanner: Accounts and Privileges **134**
FlexNet Inventory Scanner: Implementation **135**
FlexNet Inventory Scanner: Implementation on Windows 136
FlexNet Inventory Scanner: Implementation on Unix-Like Platforms 145
Customizing Searches for FlexNet Inventory Scanner 148
FlexNet Inventory Scanner Command Line 149
FlexNet Inventory Scanner: Troubleshooting Inventory **152**
6. Core Deployment: Details **155**
Core Deployment: Normal Operation **156**
Core Deployment: System Requirements **158**
Core Deployment: Accounts and Privileges **162**
Core Deployment: Implementation **162**
Core Deployment: Troubleshooting Inventory **164**
7. Common: Details **166**
Common: Child Processes Invoked by the Tracker **166**
Common: Child Processes on UNIX-Like Platforms 167
Common: Child Processes on Windows Platforms 182
Common: Supporting Mutual TLS **186**
Common: Collection from Virtual Environments **189**
Common: Gathering Inventory from Solaris Zones **192**
Common: Targeting for Solaris Zones 193
Common: License Reconciliation Considerations for Processor-Based Licenses 195
Common: Importing Registry Settings **195**
Targeting Individual Entries 196
Collecting All Uploaded Entries 198
Common: Acting on Inventory Results **199**
Common: Resolving Inventory Records **202**

Common: Ensuring Distinct Inventory 203
Common: Identifying Related Inventory 204
Common: Choosing Values from Multiple Inventory Records 215

8. Selecting Inventory Beacons 217

Overview 217
Saving the Configuration 218
Prioritizing Inventory Beacons 219
Using a Single Inventory Beacon 220
Supplied Algorithms 222
MgsADSiteMatch: Match to Active Directory Site 224
MgsBandwidth: Bandwidth Priorities 225
MgsDHCP: Retrieve location list from DHCP server options 226
MgsDomainMatch: Match to Domain Name 229
MgsIPMatch: Match to IP Address 230
MgsNameMatch: Match Prefixes of Computer Names 231
MgsPing: Server with the Fastest Response 232
MgsRandom: Random Priorities 233
MgsServersFromAD: Retrieve List from AD 234
MgsSubnetMatch: Match to Subnet 237

9. Command Lines 239

mgspolicy Command Line 239
ndlaunch Command Line 241
ndschedag Command Line 246
ndtrack Command Line 247
ndupload Command Line 254

10. Preferences 258

[Registry] Explained 258
AddClientCertificateAndKey 259
AllowedPkgTypes 260
Application 261
AutoPriority 262
CALInventory 262
CALInventoryPeriod 263
Catchup 264
CheckCertificateRevocation 265

CheckServerCertificate **266**
CommonAppDataFolder **267**
Compress (application usage component) **268**
ComputerDomain **269**
ConnectionAttempts **270**
DateTimeFormat **271**
DefaultSchedulePath **272**
Directory **272**
DisableAllAgentUploads **273**
Disabled (application usage component) **274**
Disabled (schedule component) **276**
DisablePeriod **277**
DownloadSettings **278**
EmbedFileContentDirectory **279**
EmbedFileContentExtension **281**
EmbedFileContentMaxSize **281**
ExcludeDirectory **282**
ExcludedMSIs **285**
ExcludeEmbedFileContentDirectory **285**
ExcludeExtension **286**
ExcludeFile **288**
ExcludeFileSystemType **289**
ExcludeLocalScriptRule **290**
ExcludeMD5 ..... **292**
ExecutablePath **293**
GenerateMD5 ..... **294**
Hardware **294**
HardwareChangesClassPropertyBlacklist **295**
HighestPriority **296**
Host **297**
http_proxy **298**
https_proxy **299**
IBMDB2CommandTimeoutSeconds **299**
IncludeDirectory **300**
IncludeExecutables **303**

IncludeExtension 304
IncludeFile 305
IncludeFileSystemType 306
IncludeLocalScriptRule 308
IncludeMachineInventory 310
IncludeMD5 310
IncludeNetworkDrives 311
IncludeRegistryKey 313
IncludeUserInventory 315
InstallDefaultSchedule 316
InstallProfile 317
InventoryFile 317
InventoryScriptsDir 318
InventorySettingsPath 320
InventoryType 321
LauncherCommandLine 321
LogFile (application usage component) 322
LogFile (installation component) 323
LogFile (inventory component) 324
LogFile (policy component) 325
LogFile (upload component) 325
LogFileOld (application usage component) 326
LogFileOld (installation component) 327
LogFileOld (policy component) 328
LogFileOld (upload component) 328
LogFileSize (application usage component) 329
LogFileSize (installation component) 330
LogFileSize (policy component) 331
LogFileSize (upload component) 332
LogLevel (inventory component) 333
LogLevel (policy component) 334
LogModules (application usage component) 335
LogModules (inventory component) 336
LogModules (policy component) 337
LowestPriority 337

LowProfile (application usage component) **338**
LowProfile (inventory component) **339**
MachineID **339**
MachineInventoryDirectory **341**
MachineName **342**
MachinePolicy **342**
MachinePolicyDirectory **343**
MachinePolicyPackageDirectory **344**
MachineScheduleDirectory **345**
MachineZeroTouchDirectory **346**
ManageSoftPackages **347**
ManualMapperDefaultPriority **347**
MaximumNetworkRetryPeriod **348**
MaxKeepAliveLifetime **349**
MaxKeepAliveRequests **350**
MinInventoryInterval **351**
MinRunTime **352**
MSI **353**
Name **354**
ndsensNetType **355**
NetworkHighSpeed **356**
NetworkHighUsage **357**
NetworkHighUsageLowerLimit **358**
NetworkHighUsageUpperLimit **359**
NetworkLowUsage **360**
NetworkLowUsageLowerLimit **361**
NetworkLowUsageUpperLimit **362**
NetworkMaxRate **363**
NetworkMinSpeed **364**
NetworkRetries **365**
NetworkRetryPeriodIncrement **366**
NetworkSense **367**
NetworkSpeed **368**
NetworkTimeout **369**
no_proxy **370**

OnConnect ... 370
OnlyGenerateIfHardwareChanged ... 371
OracleEnvironmentCmdTimeoutSeconds ... 372
OracleInventoryAsSysdba ... 373
OracleInventoryUser ... 374
Password ... 376
PerformDockerInventoryScan ... 377
PerformIBMDB2Inventory ... 378
PerformIBMWebSphereMQScan ... 379
PerformKvmInventory ... 380
PerformLocalScripting ... 381
PerformOracleFMWScan ... 382
PerformOracleInventory ... 384
PerformOracleJavaAuditScan ... 385
PerformOracleListenerScan ... 387
PerformSymantecSFScan ... 388
PerformVirtualBoxInventory ... 389
PlatformSpecificPackages ... 390
PolicyPackageRefreshPeriod ... 391
PolicyServerURL ... 392
Port ... 393
PreferenceUpdatePeriod ... 394
PreferIPVersion ... 395
PrioritizeRevocationChecks ... 396
Priority ... 398
Priority (manual mapper) ... 398
ProcessUpdatePeriod ... 399
ProductUpdatePeriod ... 400
ProgramFiles, ProgramFilesX86Folder, ProgramFilesX64Folder ... 401
Protocol ... 402
Recurse ... 402
Regex ... 403
RunInventoryScripts ... 404
ScheduleType ... 405
ScriptDir ... 406

SelectorAlgorithm 407
SendTCPKeepAlive 408
SessionBackupPeriod 410
ShowIcon (installation component) 410
ShowIcon (inventory component) 411
Software 412
SourceFile 413
SourceRemove 414
SSLCACertificateFile 415
SSLCACertificatePath 416
SSLClientCertificateFile 416
SSLClientPrivateKeyFile 418
SSLCRLCacheLifetime 419
SSLCRLPath 420
SSLDirectory 421
SSLOCSPCacheLifetime 421
SSLOCSPPath 422
Startup 423
StrictInstall 424
SysDirectory 425
TrackProductKey 426
UILevel 426
UIMode 426
Upload 427
UploadLocation 428
UploadPassword 430
UploadPeriod 430
UploadRule 431
UploadSettings 432
UploadType 433
UploadUser 434
UsageDirectory 435
UseAddRemove 436
UseManualMapper 437
UseMSI 437

User ... 438
UserDefinedOracleHome ... 439
UserHardware ... 441
UserInteractionLevel (installation component) ... 442
UserInteractionLevel (inventory agent) ... 443
UserInventoryDirectory ... 444
UserProcessesOnly ... 445
UserScheduleDirectory ... 445
UserZeroTouchDirectory ... 446
Version ... 447
VersionInfo ... 448
WinDirectory ... 449
WMI ... 450
WMIConfigFile ... 450

11. File Formats ... 452

Application Usage Files (.mmi) ... 452
Catalog files (.osd) ... 455
Policy Files (.npl) ... 457
Schedule Files (.nds) ... 460
WMI Configuration File (wmitrack.ini) ... 462

Part II. Two FlexNet Kubernetes Agents ... 467

1. The Lightweight Kubernetes Agent ... 470

How the Lightweight Kubernetes Agent Works ... 470
Downloading the Lightweight Kubernetes Agent ... 471
Scripted Installation ... 474
Manual Installation ... 480
Managing Certificates for TLS ... 482
Uninstalling the Lightweight Kubernetes Agent ... 484
Options for the Lightweight Kubernetes Agent ... 485

2. Inventory Uploaded by the Kubernetes Agents ... 496

Kubernetes Inventory Uploads ... 497
Inventory from IBM License Service ... 505

Part III. Collecting Inventory from Container Images ... 511

1. Container Image Inventory Tool imgtrack ... 512

Prerequisites for imgtrack **513**

Downloading the imgtrack Script **514**

2. How imgtrack Works 516

Handling the ndtrack Binary **518**

Identifying Container Images **519**

Labels for the Derived Image **521**

Options for the imgtrack Script **522**

The FlexNet Inventory Agent

After an introductory matrix comparing the results you may expect from various inventory sources, the remainder of this part is solely concerned with the collection of FlexNet inventory using the FlexNet inventory agent.

Tip: *This does not include collection of inventory from servers in Kubernetes clusters. For that functionality, see part II, covering Two FlexNet Kubernetes Agents.*

Learning about FlexNet inventory gathering involves understanding different configurations of the code elements that gather the inventory. As well, where these code elements are deployed influences both the capabilities and the management requirements of the system. In fact, even the methods of deployment can have some influence.

For these reasons, the first chapter in this part summarizes these factors to arrive at a *standard nomenclature*, used consistently throughout this document. There is also an overview of some key scenarios to help you choose which combination of code elements, deployment location, and deployment method you need.

The following chapters each treat just one of the resulting "cases". Comparable details are provided for each of the cases. The concept here is to simplify your reading. Instead of needing to tease out from mixed documentation which data point applies to your case, you need read only the one chapter that applies to your case. It contains the relevant data points exclusively for that case. (Since there is considerable overlap between the cases, this makes the document overall rather repetitive. Fortunately, you do not need to read it all. It is a reference work, not a narrative.) Notice further that each case's *Details* chapter includes topics for:

- The *Normal Operation* expected for the case — this may help you make your final choice about the method(s) of gathering FlexNet inventory that you will use in your enterprise
- The *System Requirements* specific to this case
- The *Accounts and Privileges* required for the case
- The *Implementation* used for this case (for example, how to deploy the code entities for the case, if deployment is in fact required)
- *Troubleshooting* comments applicable to the case.

Those chapters are followed by one that covers material that is common to all cases. You may choose topics from the *Common: Details* chapter selectively, if they apply to your environment.

To complete the part, there are chapters of reference material covering the command lines for key code elements, and a significant number of preference settings that can control the behavior of those code elements.

But first, we need a clear understanding and nomenclature, covered in the first chapter.

1

Understanding What, Where, How, and Why

Discussing the techniques for inventory collection with FlexNet Manager Suite is made challenging because there are two closely-related (but distinct) code entities that can be used. Each of these can be deployed to different locations within your enterprise network, and managed in different ways. How they are deployed (whether by automation within FlexNet Manager Suite or by other techniques you choose to use) also affects both management and functionality. Finally, different combinations of these factors are best suited to different scenarios of what you are trying to achieve and what aspects you want to avoid.

This section establishes a map of the terminology used throughout this reference, and the ways that different approaches affect the system requirements for, and management of, inventory tools available within FlexNet Manager Suite. (This discussion excludes inventory collected by other "third party" tools, and imported into FlexNet Manager Suite through adapters, whether built-in adapters or customized add-ons.)

Those unfamiliar with the inventory technology in FlexNet Manager Suite are encouraged to work through the following foundational topics. Experts who want only a terminology update can use the table below as an executive summary of what is deployed, how, and where; with each combination given a case name used consistently throughout this document:

Case name	What	Where	How
Adopted	FlexNet inventory agent	Target inventory device	Automatically by FlexNet Manager Suite
Agent third-party deployment	FlexNet inventory agent	Target inventory device	Third-party deployment
Zero-footprint	FlexNet inventory core components	On an inventory beacon	Not applicable (installed with the inventory beacon)
FlexNet Inventory Scanner	FlexNet inventory core components (in self-installing wrapper)	Target inventory device, or a network share	Third-party deployment

Case name	What	Where	How
Core deployment	FlexNet inventory core components	Target inventory device running Microsoft Windows	Third-party deployment

What Can Be Used for FlexNet Inventory Collection

There are two distinct code entities available for the collection of FlexNet inventory (both software and hardware inventory) by FlexNet Manager Suite within your enterprise:

- The complete, most powerful, backward-compatible entity is called the **FlexNet inventory agent**. Whenever this name is used within this document, it always means the *complete* agent. The complete FlexNet inventory agent is the entity that is deployed automatically by FlexNet Manager Suite onto target inventory devices, if you choose to allow it. Its purposes are:
  - To take inventory of both the hardware and software on a computing device, and return an XML document (`.ndi`) describing this inventory
  - By default, to return additional files for extended discovery and inventory tasks, such as taking inventory of any Oracle Database discovered on the local computer
  - To optionally track usage of applications on the same device, based on watching specified files that form part of the application.
- The smaller footprint option, with reduced functionality that covers inventory collection most simply, is called **FlexNet inventory core components**. Although this includes the same core executable (`ndtrack`) as the complete FlexNet inventory agent, we consistently use this distinct name, FlexNet inventory core components, to help clarify the reduced set of functionality and differences in deployment and management. Its sole purpose is:
  - To take inventory of both the hardware and software on a computing device, and return an XML document (`.ndi`) describing this inventory.

  Tip: *If you are deploying the FlexNet inventory core components yourself, it is possible to deploy an additional special-purpose file to add the extended discovery and inventory tasks mentioned above; but this is not included by default.*

This clear distinction between the two distinct code entities is fundamental.

Tip: *For those looking for the lightweight FlexNet Inventory Scanner, this is not a separate code entity, but an easy method of delivering the FlexNet inventory core components and cleaning them up when "our work here is done". More detail follows in later topics.*

Already we can see that, alone, the distinction between code entities is not enough to fully define the functionality set, since operational contexts also make a difference. For example, the FlexNet inventory core components are included as a standard element with each FlexNet inventory beacon.

- On one hand, this explains why inventory collection managed by the inventory beacon (remote from the target

device) *cannot* collect application usage information, in contrast to the FlexNet inventory agent which (locally installed on the target device, with additional monitoring capabilities) *can* track usage. The distinction explains the missing functionality.

- On the other hand, the FlexNet inventory core components achieve more when operating on an inventory beacon than they do if you deploy them to another file share. This is because the inventory beacon provides additional code (installed as part of FlexNet Beacon) and integration services available only in this context.

As another example, as noted above, the FlexNet inventory core components can be delivered as the FlexNet Inventory Scanner.

Therefore, a full understanding of available functionality (and management needs) requires *both* knowledge of the different code entities, *and* knowledge of contexts. The question of contexts and delivery methods is discussed shortly, in Deployment Overview: Where to, and How. But first, some more insight into the differences between these two basic code entities.

A note about 'agents'

The word "agent" can be used with different scope. For example, the FlexNet inventory agent is a scope that includes several executables performing different functions. For example, one of these is `ndupload`, which is also colloquially called the "upload *agent*". Both the large scope and the small scope of the word "agent" fit the general definition of a software "agent": a software program that runs on a computer to collect information and transfer it to a central location. However, for clarity, this document reserves the word "agent" for the larger scope of the entire FlexNet inventory agent, referring to the smaller elements as either "elements", "components", or as individual "executables".

In every case, whether invoked as part of the FlexNet inventory agent or through the FlexNet inventory core components, the executable `ndtrack` (amongst others) runs in the context (memory) of the target inventory device. For this reason, this document does not describe use of any of these approaches as "agentless". Some people like to use this term to mean that nothing is permanently installed on the target device, and there are deployment scenarios available that avoid such a permanent footprint. But the relevant code elements still execute in the context of the target machine. (Some *other* kinds of specialized inventory collection by FlexNet Manager Suite are truly agentless, in the sense that no 'agent' code element executes in the target context. Examples include an inventory beacon executing remote discovery and inventory for Oracle, Oracle VM, and VMware, which make use of services already available on the target machines. In these cases, execution is in the context of the inventory beacon, using the API offered by the installed software.)

Differences

Here are the key differences between the FlexNet inventory agent and the FlexNet inventory core components, using the Windows platform as our example (UNIX-like platforms support equivalent functionality).

Function	FlexNet inventory agent	FlexNet inventory core components
Included executables* ***Tip:*** *The full FlexNet inventory agent includes several components present for backward compatibility, so that the latest FlexNet inventory agent can function in earlier implementations during migration, for example.* Each case (FlexNet inventory agent and FlexNet inventory core components) also includes additional configuration files and libraries, here omitted for clarity.	• `ndtrack.exe` • `getSystemId.exe` • `mgspolicy.exe` • `mgssecsvc.exe` (and its plug-in `mgsusageag`) • `ndinit.exe` • `ndlaunch.exe` • `ndschedag.exe` • `ndsens.exe` • `ndtask.exe` • `ndupload.exe` • `getSystemId.exe` • `UsageTechnicianTool.exe` Still included, but now deprecated: • `mgsdl.exe` • `mgsmsilist.exe` • `mgspostpone.exe` • `reboot.exe`	• `ndtrack.exe` • `getSystemId.exe`
Inventory types	• Machine • User (Windows only, backward compatibility only)	• Machine • User (Windows command line only, backward compatibility only)
Policy, rules, and settings	Set in the web interface, automatically managed through the inventory beacons, and applied automatically as specified.	None included. Must be either: • Manually managed, usually through changing the command lines for scheduled tasks and the like • Managed by the FlexNet Beacon code (see Deployment Overview: Where to, and How for further details).

Function	FlexNet inventory agent	FlexNet inventory core components
Scheduling (inventory collection)	Built in (follows schedule set in the web interface). Can be used for high-frequency inventory gathering for IBM PVU licensing.	None. Must be scheduled using external tools (including FlexNet Beacon).
Updates	Self-updating to a version set either in the web interface, or by using a supplied command line tool.	None. Third-party deployments must be managed independently (presumably using the same deployment tools as were used in the initial deployment). (Components installed with FlexNet Beacon are also updated with future self-updates of the inventory beacon.)
Upload behavior (for collected inventory)	On by default	Off by default. To turn on requires: • Windows: Custom command line • UNIX: Custom command line or setting in `ndtrack.ini`. (On an inventory beacon, the FlexNet Beacon manages uploads.)
Usage tracking	Available, subject to configuration.	Not supported.

* The functions of some of the key executables included in the FlexNet inventory agent are as follows:

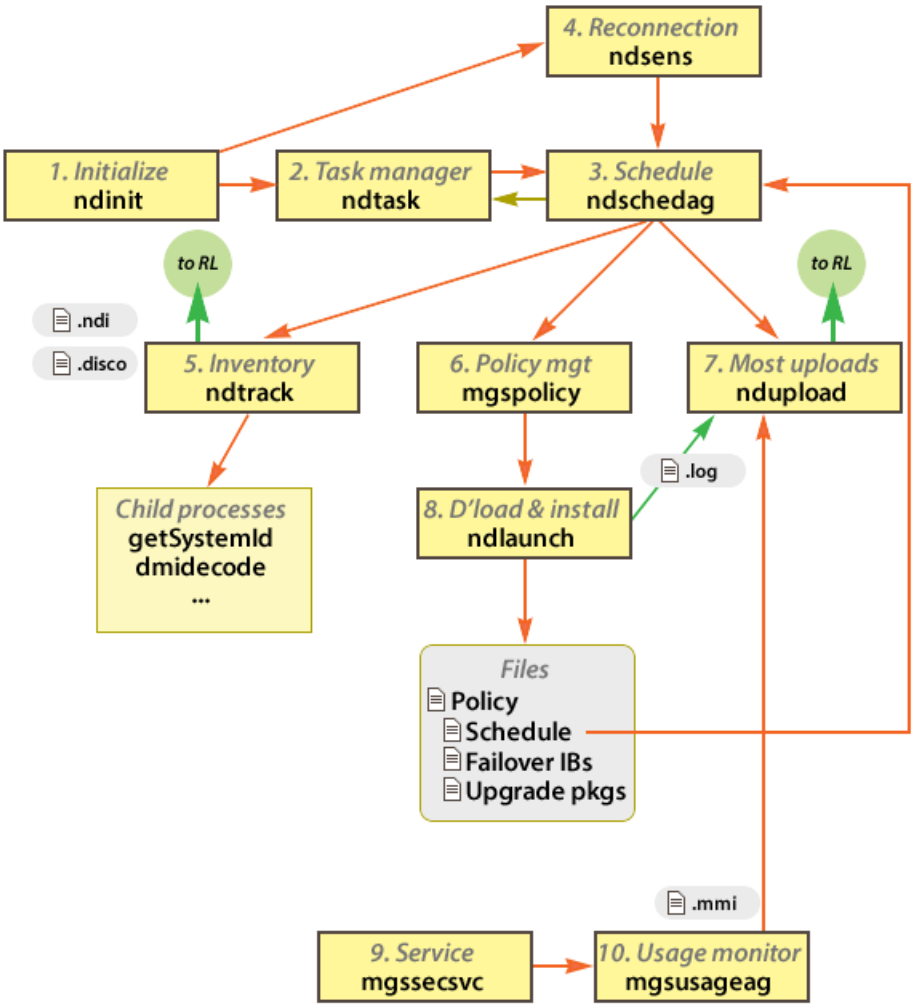
- The core component for inventory collection, `ndtrack`, which can optionally also immediately upload the gathered inventory to an inventory beacon
- The upload component (`ndupload`), which retries transfers of the inventory results to an inventory beacon to recover from transient network issues
- The schedule component (`ndschedag`) to coordinate execution of the other components
- The task management component (`ndtask`), functionally identical with `mstask` (part of the Microsoft Task Scheduler) but available across platforms
- The policy component (`mgspolicy`), responsible for managing the various rules that the control the overall FlexNet inventory agent
- The installation component (`ndlaunch`) which downloads policy, schedule, self-update and other packages required for operation
- The service component (`ndinit`, available only on Windows) is automatically initialized as a service on machine reboot, and is responsible for starting `ndtask`. On UNIX-like platforms, `ndtask` is the service, which is initialized in platform-specific ways after a reboot.

For more details about the relationships between these components, see Agent Architecture.

Agent Architecture

This architectural diagram and the following notes give more insight into the interactions between the various components of the complete FlexNet inventory agent. In the diagram:

- Each box with a heavier outline represents one of the code components of the FlexNet inventory agent, combining a heading about purpose with the name of an executable. The numbers refer to the corresponding notes below the diagram.
- Fine red arrows indicate the process of invocation: the component where the arrow starts invokes the component where the arrow ends.
- The various file types created by components are identified by their file name extensions on the Microsoft Windows platform.
- Green arrows indicate which components are responsible for uploading the files produced; but see also the notes, as methods vary.
- Where there are variations between Microsoft Windows and UNIX-like platforms, the diagram favors the Windows presentation, and differences are explained in the notes below.



1. The `ndinit` component exists only on Windows. It is automatically initialized as a service on machine reboot, and is responsible for starting `ndtask` and, on reconnection to the network, `ndsens`.
2. The `ndtask` component is functionally identical with `mstask` (part of the Microsoft Task Scheduler), but is available across platforms. On UNIX-like platforms, it runs as a service. It functions rather like a to-do list, but does not start tasks directly. Instead, it invokes the `ndschedag` component to trigger tasks when required.
3. The schedule component (`ndschedag`) has two main functions:
   - When handed a new schedule and invoked by `ndlaunch`, `ndschedag` unpacks the contents of the schedule file (see Schedule Files (.nds)) and saves the details in separate task files (`.nts`) for use by `ndtask`. It then sends an IPC message to `ndtask` to process the updated task list.

- When a specified task is triggered, `ndschedag` is always invoked (most often by `ndtask`, and occasionally on Windows by `ndsens`), and then coordinates execution of the appropriate components (in particular, inventory gathering, policy updates, and data uploads).

4. The `ndsens` component (available only on Windows) is invoked only when the network connection is reestablished for the device on which the FlexNet inventory agent is running. When, after a disconnection, the device reconnects to the network, `ndsens` next invokes `ndschedag` to check for any scheduled events with an `OnConnect` trigger type (these are normally the `Update Machine Policy`, `Update Client Settings`, and `Upload Client Files` events). There is no equivalent functionality for the FlexNet inventory agent installed on UNIX-like machines.

5. All inventory gathering and discovery work on the local device is the responsibility of the tracker, or `ndtrack`. Once triggered by `ndschedag`, this collects inventory data in `.ndi` files, and discovery data in `.disco` files. Depending on configuration, compressed archives of `.ndi` files are produced, and archives of `.disco` files may also be produced. By default (in the case of the full FlexNet inventory agent), the tracker also tries an immediate upload of gathered data files to the `ManageSoftRL` file share on its chosen inventory beacon. If configuration or some transient networking problem prevents this upload, the files are saved on the local file system, where they are subsequently picked up by the `ndupload` component. The tracker also invokes other components of the FlexNet inventory agent, such as `getSystemId` on Windows and a supplied version of `dmidecode` on certain legacy Linux platforms. For more information about operating system elements that are also invoked by the tracker, see the platform-specific listings under Common: Child Processes Invoked by the Tracker.

6. The policy component (`mgspolicy`) is responsible for managing the various rules that the control the overall FlexNet inventory agent. Historically it had a significant role (when "client-side policy" was an option); but now that all agent policy is prepared by the inventory beacon ("server-side policy"), its main role is to invoke the download and installation component (`ndlaunch`) to check for, and if necessary download, changed policy.

7. The upload component (`ndupload`) is responsible for most uploads from the local device to its chosen inventory beacon, using the same `ManageSoftRL` file share mentioned above. (When, instead, the tracker uploads discovery and inventory results, it is using a shared library of common code from `ndupload`, so that the upload functionality is identical. This integration supports uploads in other configuration of the tracker, such as in the Core deployment case and the Zero-footprint case, when the separate `ndupload` component is not available.) The uploader may be invoked directly by other components to upload files handed off through the command line; or it may be invoked by the scheduler to look in defined folders on the local device and upload all files present there. In this way, `ndupload` provides a scheduled catch-up service to upload files which the tracker failed to upload, and saved on disk instead. (And it also follows that the *absence* of the separate `ndupload` component in the Core deployment and Zero-footprint cases means that there can be no catch-up uploads after a transient failure of uploads by the tracker.)

8. The download and installation component, `ndlaunch`, is the only component that downloads files from an inventory beacon to the FlexNet inventory agent on the local device. In each case, it checks for changed content, and only downloads files that have been updated. The first file checked is the device policy for the FlexNet inventory agent. As well as summarizing the applicable rules for agent behavior, the policy file (see Policy Files (.npl)) points to a number of other resources that the launcher downloads on demand (when changes have occurred). These include:

   - The agent configuration file
   - The current version of `InventorySettings.xml`
   - The list of available inventory beacons to which the FlexNet inventory agent may upload data ('available'

inventory beacons are generally those with IIS configured for anonymous authentication)

- The upgrade packages that support self-upgrade of the installed FlexNet inventory agent
- The agent schedule, which is either of:
  - The default global schedule set for all installed agents in the web interface of FlexNet Manager Suite
  - The special schedule used for the FlexNet inventory agent on devices linked to IBM PVU licenses, when FlexNet Manager Suite is configured for 30-minute inventory updates and sub-capacity license calculations (as an alternative to ILMT).

  If `ndlaunch` downloads a revised schedule, it invokes `ndschedag` to respond appropriately.

Depending on the agent configuration and the available upgrade packages, the launcher may download a new version of the FlexNet inventory agent, and automatically install it. The launcher also saves log files for all its activities to the local disk, from where they are subsequently uploaded by `ndupload` to support status reporting on the central application server. This means that central reporting is dependent on the schedule for uploads, which is typically set to once per day, overnight.

9. Another service that runs exclusively on Microsoft Windows is `mgssecsvc` (there is no equivalent on UNIX-like platforms). Like `ndinit`, it is automatically initialized as a service on machine reboot. It exists solely as a wrapper for its child processes (which are implemented as DLLs on Windows, and so are running whenever the service is running).
10. The component that monitors application usage (when you have usage tracking configured) is `mgsusageag`, which is a library exercised by `mgssecsvc` on Windows. On UNIX-like platforms, `mgsusageag` is a service in its own right. Because of the ephemeral nature of usage data, `mgsusageag` invokes the uploader any time it has usage data (an `.mmi` file) to upload. This means that application usage data is uploaded asynchronously with relation to the upload schedule saved on the local device.

Deployment Overview: Where to, and How

Each of the FlexNet inventory agent and the FlexNet inventory core components can be deployed in different ways, and to different places within your network hierarchy. These decisions affect both the immediate deployment effort, and the ongoing management effort. In some cases, the options also impact functionality and system requirements. (These distinct names for the two entities were defined in What Can Be Used for FlexNet Inventory Collection.)

FlexNet inventory agent

FlexNet inventory agent must always be installed on the target inventory device. There are two main deployment options:

- Automatic deployment through FlexNet Manager Suite, managed by the inventory beacons, in accordance with the targets you declare in the web interface. This process is called "adoption", since it 'adopts' the target device into a closely managed environment.
- Deployment that you manage, using your existing tools and infrastructure. For convenience, we label these approaches "third-party deployment", meaning that you likely use tools/methods from a company other than Flexera. Installable packages are available through the web interface of FlexNet Manager Suite for use in third-party deployment. Under this loose heading, we include, for example:

- Deployment with a tool such as Microsoft Endpoint Configuration Manager (previously Microsoft SCCM) or Symantec IT Management Suite (formerly Altiris)
- Pre-installation on the gold image for new device configuration
- Logon scripts used in conjunction with domain controller(s)
- Active Directory Group Policy Objects
- Manual installation by a local administrator on the target device.

Provided that, in your deployment process, you identify a 'bootstrap' inventory beacon from which the FlexNet inventory agent can collect its initial policy, schedule, actions and so on, there is no significant difference in outcomes of these two methods (although you may want to configure your targets and rules differently; and for adoption, there is additional reporting and troubleshooting information available in the web interface).

FlexNet inventory core components

There are three ways that the FlexNet inventory core components are available for use within your enterprise:

- On inventory beacons (for zero footprint inventory collection)
- In a self-extracting executable format (separately named as the FlexNet Inventory Scanner)
- As a 'code folder' available for third-party deployment.

Each of these ways is further explained below.

FlexNet inventory core components on inventory beacons (zero footprint inventory collection)

The FlexNet inventory core components are always installed as part of each inventory beacon (no separate deployment or installation is required). As discussed in What Can Be Used for FlexNet Inventory Collection, the fact that it is the FlexNet inventory core components explains why usage tracking is not available from inventory beacons (usage tracking is only available in the full FlexNet inventory agent).

However, its co-location on the inventory beacon provides a special use case, because the FlexNet Beacon code provides some of the functionality normally found only in the full FlexNet inventory agent. For example, provided that the target device is not included in any target configured for adoption, FlexNet Beacon invokes the FlexNet inventory core components in line with the rules you declare in the web interface for FlexNet Manager Suite, and honors the targeting and schedules used in those rules (a level of integration that is not available to the FlexNet inventory core components in any other context). FlexNet Beacon also manages uploads of the collected data; and [self-]updates to FlexNet Beacon include the latest version of FlexNet inventory core components.

Perhaps the most significant additional functionality provided by the inventory beacon is the ability to remotely install, execute, and subsequently remove the FlexNet inventory core components on a target inventory device. Since there is no permanent agent installation on the target device either side of the inventory collection event, this model can be called "zero footprint" inventory collection. (It has previously been called 'remote execution' and 'zero touch', both of which are deprecated because of resulting misunderstandings.) This method is completely controlled by the inventory beacon, in accordance with the targeting and rules you declare in the web interface; but the code execution still occurs in the context of the target inventory device. The method details are different on different platforms:

- On Microsoft Windows, the inventory beacon creates a service on the target inventory device. This service then invokes the FlexNet inventory core components installed on the inventory beacon (using the inventory beacon as a

file share), with command-line options to cause an immediate upload of the collected data to the inventory beacon. Finally, the service removes itself, leaving "zero footprint" after the inventory collection process is completed.

- On UNIX-like platforms (which includes various UNIX varieties and OS X), the inventory beacon connects to the target device (using `ssh`), and copies (`scp`) the FlexNet inventory core components to the target device, executing them there and uploading the resulting data. It then removes the copied files, and logs out.

Obviously, these different methods impose different requirements on the various platforms, all of which are detailed in later topics. For the moment, it is enough to understand that the zero footprint inventory collection is the primary reason for the inclusion of the FlexNet inventory core components on each inventory beacon.

If you intend to collect inventory from your FlexNet Manager Suite application server(s), they must be treated as target device(s) in either of the following ways:

- Install FlexNet inventory agent on each application server and configure it to transfer the inventory file(s) to a separate standalone server before uploading it back to the central application server.
- Use remote inventory collection, known as zero-footprint, by targeting application server(s). This method temporarily installs an agent on the target device(s) and removes the agent afterward collecting and uploading inventory.

Likewise, if you intend to collect inventory from your inventory beacon, it must be treated as a target device and you must use remote inventory collection, known as zero-footprint, by targeting inventory beacon.

Self-extracting executable (the FlexNet Inventory Scanner)

The FlexNet inventory core components are also available as a self-extracting executable. This format has been called the lightweight FlexNet Inventory Scanner ('lightweight' because of its relative ease of deployment and execution). For Microsoft Windows, this a separate executable (`FlexNetInventoryScanner.exe`), and on UNIX-like platforms, it is wrapped as a shell script (`ndtrack.sh`). If you copy the FlexNet Inventory Scanner to a target device (or for Windows, to a share accessible by target devices), and execute it with optional command-line preferences (or even by double-clicking the Windows EXE, for test purposes), the following actions occur:

- The executable extracts the FlexNet inventory core components (on UNIX-like system, it first determines the current operating system, and then writes the files appropriate to that platform).
- The `ndtrack` inventory component is immediately executed. If you provided any command line options, these are passed directly through to `ndtrack`.
- If your command line included an option for an upload location, the resulting data is uploaded; and if not, the data file(s) are saved locally for your inspection and management.
- All extracted FlexNet inventory core components are then removed (only the FlexNet Inventory Scanner is left, in the folder where you had copied it originally).

The process, then, is not greatly different from the zero footprint inventory collection driven by the inventory beacon. The main differences are:

- The FlexNet Inventory Scanner is independent of any inventory beacon, so that you control its deployment, updating, and so on.
- The FlexNet Inventory Scanner does not respond to schedules or rules set in the web interface of FlexNet Manager Suite, so you control its operational behaviors, uploads, and so on.

- On Windows, the FlexNet inventory core components are extracted on the target device and later deleted, rather than being run from a file share as in the Zero-footprint case. (On UNIX-like systems, both approaches temporarily save the appropriate executable on the target device.)
- The FlexNet Inventory Scanner remains in the location where you placed it, so that there is a small disk footprint (documented later).
- By default, the FlexNet Inventory Scanner has less specialized functionality than the zero footprint inventory collection managed by the inventory beacon (and, of course, less than the complete FlexNet inventory agent). FlexNet Inventory Scanner cannot collect Oracle Database inventory, for example, or some details about Microsoft SQL Server. However, as described later, you can enable this specialized additional functionality by deploying an auxiliary `InventorySettings.xml` control file.

The FlexNet Inventory Scanner meets the requirement (particularly for Windows) of a single executable that can be copied or shared, and *just run*. However, it incurs the overhead of installing the FlexNet inventory core components each time it is run (or on UNIX, writing the appropriate files per platform). If you wish to avoid that overhead, we have the third deployment option.

Code folder for third-party deployment

This approach is conceptually very simple. You simply take the folder of FlexNet inventory core components, and use your preferred method to deploy this. It is feasible to deploy it directly onto target inventory devices running Windows, but impractical (and unsupported) for the complexities of UNIX-like environments. As before, your deployment options for Windows include:

- Deployment with a tool such as Microsoft Endpoint Configuration Manager (previously Microsoft SCCM) or Symantec IT Management Suite (formerly Altiris)
- Pre-installation on the gold image for new device configuration
- Logon scripts used in conjunction with domain controller(s)
- Active Directory Group Policy Objects
- Manual installation by a local administrator on the target device.

When you use this approach, you take responsibility for management (such as version control, updates and compatibility) and operations (such as scheduling, command line options, and uploads). If you require advanced inventory functionality, you must also deploy and manage the `InventorySettings.xml` file.

This option is simply referred to as Core deployment, meaning third-party deployment of the FlexNet inventory core components.

Summary

This discussion leads us to the following matrix of what is deployed, how, and where, with each combination given a unique case name. Subsequent topics provide more details for each of these cases.

Case name	What	Where	How
Adopted	FlexNet inventory agent	Target inventory device	Automatically by FlexNet Manager Suite

Case name	What	Where	How
Agent third-party deployment	FlexNet inventory agent	Target inventory device	Third-party deployment
Zero-footprint	FlexNet inventory core components	On an inventory beacon	Not applicable (installed with the inventory beacon)
FlexNet Inventory Scanner	FlexNet inventory core components (in self-installing wrapper)	Target inventory device, or a network share	Third-party deployment
Core deployment	FlexNet inventory core components	Target inventory device running Microsoft Windows	Third-party deployment

Typical Scenarios and Use Cases

Here are some typical use cases for the various combinations of code objects and deployment approaches. This high-level summary helps you decide which approach you wish to adopt. Subsequent topics provide in depth coverage of each approach. By settling on your approach as early as possible, you can minimize study time.

The bold keywords in the following use cases are the ones summarized at the end of Deployment Overview: Where to, and How.

A replacement for ILMT managing IBM PVU licenses

"By agreement with IBM, we will use FlexNet inventory tools for sub-capacity reporting on IBM PVU licenses."

It is mandatory to use the full FlexNet inventory agent, locally installed on target inventory devices, whether in the **Adopted** case or the **Agent third-party deployment** case.

Minimum on-going management

"I have a straight-forward network configuration, and I am OK to provide a domain administrator password for installation. Thereafter, I want automatic self-updating of the system to version limits that I control, and maximum automation of all processes — including managing the mobile devices of our road warriors."

Consider the **Adopted** case.

Minimum deployment effort, minimum management

"We have well-established deployment processes. I don't want another one, and I don't want to store passwords, whether for deployment or operation. I also need to track usage to help manage license harvesting and redeployment. I must be able to control where the agent is installed. And I also want maximum automation for future management."

Consider **Agent third-party deployment**. After you manage the initial deployment, the FlexNet inventory agent self-manages in line with your policies set in the web interface of FlexNet Manager Suite. And unlike zero footprint inventory collection, there is no on-going requirement for password storage.

A quick test

"I want to run the core components in a test environment and inspect the inventory that is returned. I won't be tracking usage."

Consider **FlexNet Inventory Scanner**. Easy to deploy, test with default settings, and remove cleanly afterward.

Specialized and focused

"I have a moderately-sized group of servers from which I need to collect specialized inventory, such as Hyper-V, VMware, and I also have some Oracle virtual machines."

Consider **Zero-footprint** inventory collection. The inventory beacon can control agentless inventory collection for several specialized inventory types, as well as agent-based general hardware and software inventory when required; all conveniently controlled through the definition of rules in the web interface for FlexNet Manager Suite.

Leveraging current infrastructure

"We have deployment, monitoring, and security under tight control. We want minimum disruption of our current practices."

For Windows platforms, consider **Core deployment**, when it is necessary to augment your current inventory-gathering tools and methods with the advanced inventory-gathering capabilities of FlexNet Manager Suite. (You can, of course, import inventory from other tools, but that is not the subject of this document.) With this option, you manage deployment as well as future updates, scheduling, monitoring and any required remediation, and so on.

On UNIX-like platforms, consider using **FlexNet Inventory Scanner**. Once again, with this approach there are no downloads or self-updates, so that you manage deployment as well as future updates, scheduling, monitoring and any required remediation. When you execute the `ndtrack.sh` shell script, it sets environment variables, selects the platform-specific version of the `ndtrack` executable and writes it to disk, executes it (passing in any command-line parameters you specify), and then removes the executable. More details are in FlexNet Inventory Scanner: Operation on UNIX-Like Platforms.

Support for IPv6 Networks

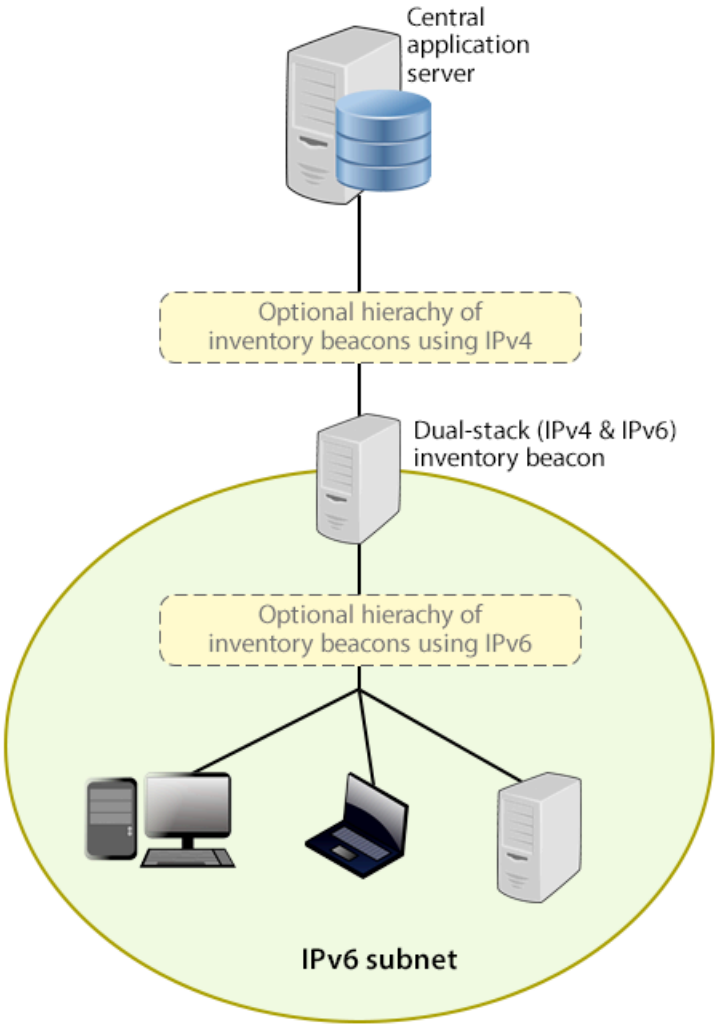
FlexNet Manager Suite supports inventory collection (and limited discovery in the case of local collection of Oracle data) within IPv6 networks. However, current requirements for, and restrictions on, this functionality mean that only the following forms of gathering FlexNet inventory are supported on IPv6 zones at the 2023 R2 release:

- Agent third-party deployment
- FlexNet Inventory Scanner
- Core deployment (where you use third-party technology to deploy just the core inventory executables).

Specifically, since no forms of remote execution from an inventory beacon are currently supported in an IPv6 network, automatic deployment of the FlexNet inventory agent is not possible (that is, the Adopted case is impossible); and all forms of Zero-footprint activity are excluded in IPv6 networks. Of course, these remain available in IPv4 networks, along with all other existing functionality.

There are two slightly different approaches to support IPv6 in your on-premises implementation. The first of these is best suited to the case where you have limited subnets running IPv6, while much of your network continues using the

IPv4 address family. In this case, it is simplest to place your application server(s) in a subnet using IPv4, so communications with the central server(s) use either HTTP or HTTPS communications running over an IPv4 network protocol.The need to support the IPv4 protocol at the top level of the architecture, and the IPv6 protocol at the low level with the local FlexNet inventory agent, means that at least one inventory beacon must be a dual-stack server that provides the bridge between the two protocols, as shown in the following architectural sketch:



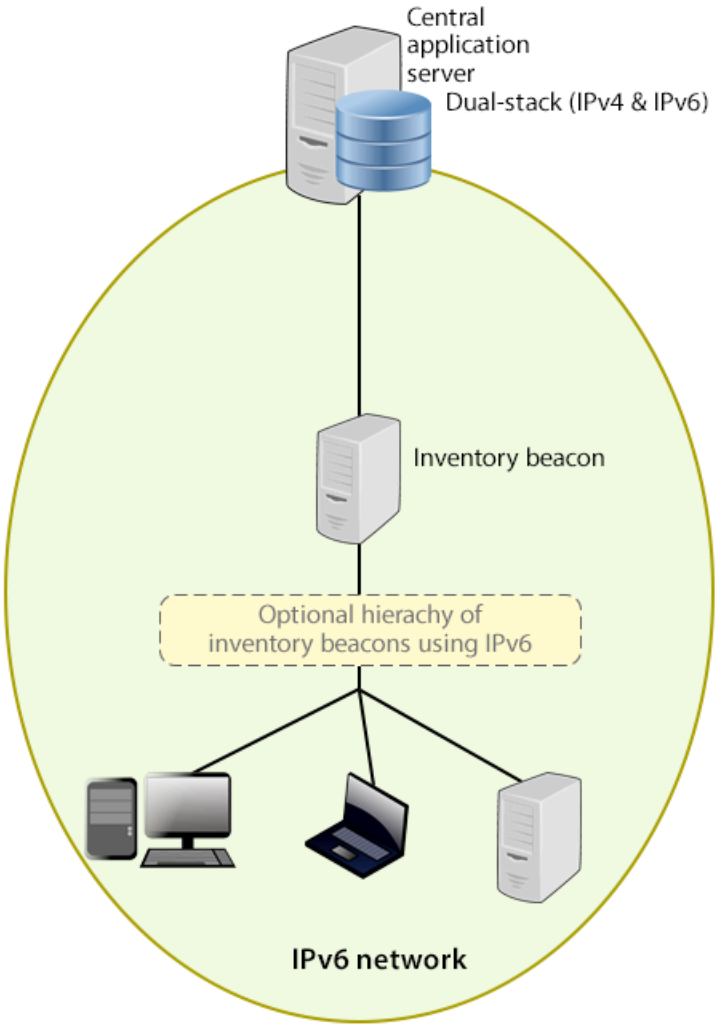
Reading from top to bottom, this sketch shows:

- Your application server (or in larger implementations, multiple servers) continue(s) to support HTTP or HTTPS communications over an IPv4 network layer.
- Within IPv4 zones of your network, you may deploy as many inventory beacons as required, either as a flat layer where each communicates directly with the application server, or in a hierarchy, as dictated by your network requirements. Of course, these inventory beacons provide full functionality, supporting all forms of FlexNet inventory gathering from target inventory devices within the IPv4 network (for simplicity, these devices in the IPv4 zone are not shown in the sketch above).
- At least one inventory beacon must be a dual stack device that supports both IPv4 and IPv6 network layers. It does not matter whether this is achieved using two Network Interface Cards (NICs) or a single configurable NIC. The IPv4

interface links upward to its parent (whether that be to another inventory beacon in the hierarchy or directly to the application server). The IPv6 interface links downward to those of its child devices that are in the IPv6 zone (of course, other devices in the IPv4 network could also communicate through this inventory beacon, given its dual stack architecture). As shown, these IPv6 children may optionally include a further hierarchy of inventory beacons (which child inventory beacons would then be operating entirely within the IPv6 network).

- Eventually, target inventory devices within the IPv6 zone that have locally installed FlexNet inventory agents communicate with at least one inventory beacon in the same zone; or where the lightweight FlexNet Inventory Scanner has been run on a target device, this can also communicate with the inventory beacon.

A variation on this approach is possible for cases where the majority of your network uses the IPv6 address family (and in particular, all your target inventory devices are in IPv6 subnets). You can make your central application server(s) dual-stack machines, supporting IPv4 for communications between themselves (this much is mandatory); but have *all* communications with inventory beacons using the IPv6 address family. This produces the following variation on our architecture diagram:



There are further restrictions and requirements to add to these general sketches:

- All inventory beacons operating within an IPv6 network (whether as single-stack IPv6 devices or dual-stack IPv4 and

IPv6 devices) must utilize Microsoft IIS as the web service. The simple alternative self-hosted web server does not support the IPv6 protocol.

- Inside an IPv6 network, an inventory beacon cannot import Active Directory details. However, a dual-stack inventory beacon that can communicate with a domain name server (DNS) over IPv4 can still import Active Directory data. Alternatively, an inventory beacon *co-installed on your central application server* (which by definition must have IPv4 available to it) can still access a DNS on IPv4 and import Active Directory data.
- Inside an IPv6 network, an inventory beacon cannot do any of the following:
  - Import inventory from third-party sources
  - Import business data from other systems (such as your purchasing or HR systems)
  - Communicate with SAP systems in your IPv6 environment
  - Perform any inventory beacon-based discovery or remote inventory collection across the IPv6 subnet, including VMware host scans (such as required for special 30-minute scans for IBM PVU license management)
  - Adopt target inventory devices that can communicate only on an IPv6 subnet (instead, use third-party deployment to install the FlexNet inventory agent on target devices within an IPv6-only subnet).

  However, once again, a dual-stack inventory beacon that can communicate with a DNS over IPv4, and contact the various sources also exclusively over IPv4, still supports all the above functionality on the IPv4 side. This is also true of an inventory beacon co-installed on the application server.

2

Adopted: Details

This chapter provides great detail about the FlexNet inventory agent automatically deployed through the inventory beacons to target devices (labeled the Adopted case in Deployment Overview: Where to, and How).

This document provides a consistent set of data (as far as possible) across all the different use cases, each in its own chapter. This means that, once you have chosen your preferred use case, you can focus only on the details for that one, and ignore all other use case chapters.

Because FlexNet inventory collection supports a range of operating system platforms, some topics necessarily contain several sets of information, and you can select only those details that apply to your chosen platform(s).

In addition to the distinct chapters for the different use cases, you should also review the subsequent chapter on functionality that is common throughout. This is followed by detailed reference material on command lines, preferences, file formats, and the like.

Adopted: Normal Operation

The operating procedures for the full FlexNet inventory agent deployed automatically through adoption are consistent across Microsoft Windows and UNIX-like systems, with some obvious distinctions in system dependencies like file paths. This topic assumes that adoption is complete (for those details, see Adopted: Implementation), and covers operations thereafter. The entire process is covered, including what happens to the collected data after it is uploaded by the FlexNet inventory agent. Each numbered step provides a summary point, followed by further specific details that you can skip over until needed.

Important: *Adoption is not currently supported in IPv6 networks. However, after installation, operation of the FlexNet inventory agent within an IPv6 network is supported. Therefore, for IPv6 networks, see Agent Third-Party Deployment: Details and its following topics.*

1. Normally only once (probably even before adoption, although you can certainly change it later if required), you set the schedule for operations of the FlexNet inventory agent in the web interface of FlexNet Manager Suite (navigate to **Discovery & Inventory > Settings**, in the **Inventory agent schedule** section).

   This schedule is automatically downloaded to all inventory beacons for delivery to installed FlexNet inventory agents on managed devices.

> ***Tip:*** *Once discovery and adoption are completed, on-going inventory operations of the installed FlexNet inventory agent in the Adopted case are not controlled by inventory rules created in the web interface. This includes the schedules that form part of rules, which have no effect on post-adoption inventory gathering by the installed FlexNet inventory agent (which is a state-based device, self-managing to align with its policy). It is for this reason that the schedule for on-going inventory operations in the Adopted case are set as described above, rather than as part of discovery and inventory rules.*

2. Components of the FlexNet inventory agent are triggered by a long-running process (`ndinit` on Windows, and `ndtask` on UNIX-like platforms). The process is restarted automatically after a machine reboot.

3. Immediately upon first installation, and thereafter at a random time once every 12 hours, each FlexNet inventory agent asks an inventory beacon for its policy.

   (For details of the policy file, see Policy Files (.npl).) The FlexNet inventory agent downloads and checks the package files that are linked in the downloaded policy. The package files are very small and quick to download and process, and each identifies the version of its related content file. If the FlexNet inventory agent determines that no content files have changed since they were last collected, no further downloads take place at this time. If anything has changed, the changed content is downloaded and the appropriate settings on the inventory device are updated. These potential downloads include any change to the operational schedule for inventory collection.

4. If usage tracking is in policy for this device, the usage component monitors the running processes on the system, recording the number of times, and for how long, each process is run.

   This component does not have any noticeable impact on system performance. (Each application being tracked adds about 250 bytes of compressed data to the upload packages.) It looks up the OS-standard installation data (such as MSI on Windows, RPM on Linux, and so on) to associate a process with the installation evidence and file paths. Technically, the usage component `mgsusageag` is a daemon on UNIX-like systems, and on Windows it's a plug-in to `mgssecsvc.exe`, which starts as a Windows service at system start-up (for further details, refer back to Agent Architecture). After the results are uploaded, usage can be reported only against applications for which there is an independent installation record. (Usage results are not used to create an installation record, since the application may have been removed within the time window where usage is tracked.)

5. Apart from usage tracking, the FlexNet inventory agent sits dormant on the target inventory device until the scheduled time for inventory collection.

   The scheduled to-do list is managed by the `ndtask` component (this is a cross-platform component matching the functionality of `mstask`). When a schedule trigger fires, `ndtask` runs `ndschedag` (passing it a GUID for the required action), and for inventory collection `ndschedag` runs the `ndtrack` component. Notice that `ndschedag` provides its own logging in `scheduler.log`, in the following default directories (there are several `LogFile` preferences that can override these default locations):

Windows platforms	`$(TempDirectory)\ManageSoft\`
UNIX-like platforms	`/var/opt/managesoft/log`

6. The `ndtrack` executable by default runs at low priority to collect software and hardware inventory details on the local device.

   This means that higher priority tasks are not interrupted, so that there is minimal impact on system performance. Inventory details are saved in an `.ndi` file on the local file system on the inventory device, by default:

Windows platforms	`$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Tracker\Inventories`
UNIX-like platforms	`/var/opt/managesoft/tracker/inventories`

This location may be altered with the `MachineInventoryDirectory` preference. This directory preserves the local copy of the inventory file (where you can inspect its contents) until it is over-written at the next inventory collection by the same account (since inventory file naming reflects the account running the inventory collection).

At the same time, a compressed (`.ndi.gz`) copy of the file is also saved, ready for upload, in a separate directory:

Windows platforms	`$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Common\Uploads\Inventories`
UNIX-like platforms	`/var/opt/managesoft/uploads/Inventories`

If the FlexNet inventory agent has uncovered any Oracle services running on the local device (and, for UNIX-like target devices that have the FlexNet inventory agent running in the full privilege default operation mode, only when the FlexNet inventory agent is running as `root`), a second `.ndi.gz` file of Oracle inventory is also generated, and saved in the same directory (an uncompressed version is not saved). As well, the Oracle discovery is reported in an uncompressed `.disco` file, saved in the `Discovery` directory (a peer of the `Inventories` directory in the uploads set). Since the behavior of the installed FlexNet inventory agent is not controlled by inventory rules set in the web interface, this Oracle discovery and inventory does not rely on those rules, and will occur even when no rules for Oracle inventory collection exist, provided that:

- `InventorySettings.xml` is available to the FlexNet inventory agent (in the folder identified in the `InventorySettingsPath` preference setting, which defaults on Windows to `$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Tracker\InventorySettings\` and on UNIX-like platforms to `/var/opt/managesoft/tracker/inventorysettings` — if either the preference or the file is missing, Oracle inventory is skipped)
- For UNIX-like target devices, the FlexNet inventory agent is running as `root` (for the installed FlexNet inventory agent, all Oracle discovery and database inventory gathering are blocked for any non-`root` account).

Tip: *A system trace on the UNIX version of `ndtrack` shows that it reads `/etc/passwd`. The UNIX `ndtrack` uses the `setpwent()` and `getpwent()` library calls to obtain the `pw_name`, `pw_dir` and `pw_shell` properties for each user. In particular, `ntrack` uses the `pw_dir` property (each user's home directory) to find user-based installation evidence for BEA and InstallAnywhere installation technologies.*

7. After collecting the hardware and software inventory, `ndtrack` immediately attempts to transfer the compressed inventory file(s) to an inventory beacon.

   The upload is a background process that does not take priority away from other current tasks running on the inventory device. The destination for the upload is `ManageSoftRL` (a web service on the inventory beacon), which on free-standing inventory beacons saves the `.ndi.gz` file(s) to the folder *`%CommonAppData%`*`\Flexera Software\Incoming\Inventories` on the inventory beacon.

Tip: *If the inventory beacon is co-located on your central application server (or in large-scale systems, the*

batch server), the `ManageSoftRL` web service does not save to disk, but instead saves directly to the database as described in step 9.

(The upload functionality is shared between the `ndtrack` and `ndupload` components. Because the upload here is attempted as part of inventory collection, upload logging for this event is in the `ndtrack` or `tracker` logs, in the path given in step 5.)

- If the initial upload is unsuccessful for any reason, there is a catch-up task on the inventory device that triggers a retry of the upload. (If there are no files awaiting upload at catch-up time, this process shuts down immediately.)
  - *Timing:* The catchup schedule is internal (downloaded to the FlexNet inventory agent in the `.nds` schedule referenced in policy), and cannot be modified in any user interface. The task is called `Upload Client Files` and occurs once daily within a one hour window, the start time for which is randomized by the inventory beacon when it regenerates the schedule file after each update of beacon policy (that is, after each change to rules or schedules in the web interface for FlexNet Manager Suite). If the inventory device is turned off at the time, there is also a catch-up on machine restart.
  - *Logging:* For this catch-up, the upload uses the `ndupload` component, so that logging for the catchup attempt is in the `ndupload` logs.
- Once the upload is successful (either originally or in catch-up), the copy of each compressed inventory file in the `...\Uploads\Inventories` folder on the inventory device is deleted (meaning that the absence of files after the upload is a sign of success, and the continued presence of files after upload attempts is a sign of failure, with stale `.ndi.gz` files overwritten at the next inventory collection). After success, the process continues below.

***Tip:** For ongoing operation of the full FlexNet inventory agent in the Adopted case, it is not required that the managed device is in a subnet assigned to the inventory beacon. (For the different requirements during discovery and adoption, see Adopted: Implementation.) At installation time, the adopting inventory beacon normally sets itself as the 'bootstrap' inventory beacon for the download of initial policy. Thereafter, the fail-over settings (linked in the downloaded policy) define every available inventory beacon (those with IIS configured for anonymous authentication), and the FlexNet inventory agent may contact the most appropriate one (by default, this is one in the same Active Directory site with the fastest ping response time at each upload time; but if these conditions cannot be met, it may be a random choice). Any inventory beacon will respond to a query from any installed FlexNet inventory agent (and calculate and provide a policy file for it), provided that the system is not in migration mode. Migration mode is set in the web interface for FlexNet Manager Suite (navigate to **Discovery & Inventory > Settings > Beacon settings > Migration mode: Restrict inventory settings to targeted devices**).*

8. FlexNet Beacon (the code entity on the inventory beacon) uploads the inventory data to its parent on a schedule set by the Microsoft Scheduled Task `Upload FlexNet logs and inventories` (by default, repeating every minute throughout the day).

   The checking cycle when the folder is empty is very quick and does not perceptibly load the inventory beacon, even though it is frequently repeated. The parent of an inventory beacon may be the central application server, or another inventory beacon if these have been arranged in a hierarchy. In the latter case, each inventory beacon in turn repeats the upload process until the data reaches the application server.

9. On the application server (or, in a scaled-up system with separate servers, the inventory server), the web service

`ManageSoftRL` receives the uploaded packages for both inventory and (if configured) usage tracking (and other uploaded files).

These are processed immediately, being loaded into the internal operations databases: inventory (`.ndi`) and usage (`.mmi`) files are loaded into the inventory database; any Oracle discovery (`.disco`) file is loaded into the compliance database. If the service gets overloaded, it will temporarily spool incoming files to its local *%CommonAppData%*`\Flexera Software\Incoming\Inventories` directory (or the peer `Discovery` folder for any Oracle discovery file). From these folders, file import is resumed under the control of Microsoft scheduled tasks (for example, `Import inventories`, which is triggered every 10 minutes).

10. On the next inventory import and license consumption calculation, the inventory and usage data is collected from the inventory database, socialized as necessary, and imported into the compliance database. Here it is used in license calculations, and made available in management views and reports.

    This import step can be triggered in one of three ways:

    - Normally, the batch scheduler triggers an import daily (by default, at 2am local time on your application server), with the license consumption calculation triggered thereafter. This default time is configurable by editing the Microsoft scheduled task `Inventory import and license reconcile` on your application server (or, in larger implementations, batch server).
    - An operator in the `Administrator` role can choose to import the waiting inventory and trigger license consumption calculation, or reconciliation, as soon as possible (navigate to **License Compliance > Reconcile**).
    - For testing, a knowledgeable system administrator could use a command line on your application server (or, in a scaled-up system, your batch server) like:

      ```
      BatchProcessTask.exe run InventoryImport
      ```

      (for details, see the *Server Scheduling* chapter in *FlexNet Manager Suite System Reference*).

Adopted: Services on UNIX

The installers for FlexNet inventory agent on UNIX-like platforms install two services which are automatically started after a successful installation and on every system boot. These services are:

- `ndtask` — The agent's task launcher. This runs `ndschedag` which executes further command lines (for example, for `ndtrack`, `ndupload`, `ndlaunch`) to perform the required actions.
- `mgsusageag` — The usage tracking agent.

The services are managed using the operating-system-specific service management tools. For example, to start `ndtask`:

Platform	Command
AIX (see note)	`/usr/bin/startsrc -s ndtask`
HP-UX	`/sbin/init.d/ndtask start`
Linux	`/etc/init.d/ndtask start`

Platform	Command
Mac OS X	`/sbin/launchctl start com.flexerasoftware.ndtask`
Solaris	`/etc/init.d/ndtask start`

Similarly, to stop the `ndtask` service:

Platform	Command
AIX	`/usr/bin/stopsrc -s ndtask`
HP-UX	`/sbin/init.d/ndtask stop`
Linux	`/etc/init.d/ndtask stop`
Mac OS X	`/sbin/launchctl stop com.flexerasoftware.ndtask`
Solaris	`/etc/init.d/ndtask stop`

The `mgsusageag` service can be started and stopped in exactly the same way.

Note: *For AIX, the agents are in a group called `managesoft` and can both be started using `startsrc -g managesoft` and stopped using `stopsrc -g managesoft`.*

Adopted: System Requirements

The following details apply to the full FlexNet inventory agent when deployed through an inventory beacon to a target device.

Supported platforms

The FlexNet inventory agent operates on the following platforms (inventory targets):

Microsoft Windows	UNIX-like platforms

• Windows Server 2008 R2 x64, 2012, 2012 R2, 2012 R2 SP1, 2016, 2019, 2022
• Windows Server Core 2008, 2008 R2 x64, 2012, 2012 R2
• Windows Server Standard (previously known as Windows Server Core) 2016, 2019
• Windows Vista, 7, 8, 10, 11
 	
 Note: *Windows systems that can run on ARM-based devices are also supported.*
	• AIX 7.1 LPARs with Technology Level 5 or later, AIX 7.2
	• Amazon Linux 2, 2023 (ARM64/AArch64; x86, 32-bit and 64-bit)
	• CentOS 6-7.5 (x86, 32-bit and 64-bit); 7.6-9 (x86 64-bit only)
	• Debian Linux 8–11.3 (x86, 32-bit and 64-bit); 11.5, 12.0 (x86 64-bit only)
	
	 Note: *For Debian Linux 9.4, 9.5 and 10 (both 32-bit and 64-bit), minimal installations of the OS core omit the `ifconfig` command. This prevents collection of the IP address and MAC address in inventory. The `root` user can use the following command to restore this functionality:*
	`apt-get install net-tools -y`
	• Fedora Linux 26 (x86, 32-bit and 64-bit); 27-38 (x86 64-bit only)
	• HP-UX 11i v3 (Itanium only), vPars/nPars
	• macOS 10.15.4–14 (applicable for both Intel and Apple M-series processors)
	
	 Note: *If you need to run the FlexNet inventory agent of version 20.1 or earlier on an Apple M-series processor ("Apple silicon"), Rosetta 2 must be installed and running. This is Apple's solution for transitioning most Intel-based applications to run on Apple silicon. There are two possible command formats for installing Rosetta 2:*
	◦ *Interactive installation that asks for agreement to the Rosetta 2 license:*
	`/usr/sbin/softwareupdate --install-rosetta`
	◦ *Non-interactive installation:*
	`/usr/sbin/softwareupdate --install-rosetta --agree-to-license`

Microsoft Windows	UNIX-like platforms
	• Nutanix AHV hypervisor ***Note:*** *All versions that are within their product support lifecycle and use the* `libvirt` *library are supported.* • OpenSUSE Leap 42.2, 42.3 (x86, 32-bit and 64-bit); 15-15.4 (x86 64-bit only) • Oracle Linux 6.0–6.10 (x86, 32-bit and 64-bit); 7.0-8.7 and 9.0-9.1 (x86 64-bit only) • Photon OS 3.0-5.0 • Red Hat Enterprise Linux (RHEL) 6.0-6.10 (x86, 32-bit and 64-bit); 7.0-8.8 and 9.0-9.2 (x86 64-bit only) • Rocky Linux 8, 9 • Solaris 10–11 (SPARC), Zones for versions 10–11 • Solaris 10–11.4 (x86), Zones for versions 10–11 • SUSE Linux Enterprise Server 12 SP3, 12 SP4, 12 SP5, 15, 15 SP1, 15.2-15.5 (x86 64-bit only) • Ubuntu 14–17.04 (x86, 32-bit and 64-bit); 17.10-23.04 (x86 64-bit only) Linux on IBM zSystems platforms are also supported. The following Linux distributions are certified for usage on the IBM zSystems architecture. For details about supported versions on different hardware types, see https://www.ibm.com/support/pages/linux-ibm-z-tested-platforms. • Red Hat Enterprise Linux • SUSE Linux Enterprise Server • Ubuntu

Disk space requirements

The following table gives three disk space figures for installation of FlexNet inventory agent:

- **Package type** gives the kind of installer package provided for each platform, and **Package size** defines the disk space to download or copy the installer for each of the platforms, before installation.
- **Installed** defines the disk space for the binary files after installation. The default locations for installation are:
  - Windows: `%ProgramFiles(x86)%\ManageSoft` (on modern 64-bit systems, this expands to `C:\Program`

`Files (x86)\ManageSoft`)

- UNIX-like systems: `/opt/managesoft`

- **Workspace** approximates a typical operating space requirement. The precise requirements depends on the self-update installer package size, the number of inventory files awaiting upload, usage tracking, the growth of log files, and the like. The default locations for this requirement are:
  - Windows: `%ProgramData%\ManageSoft Corp` for data; and `%temp%\ManageSoft` for log files. The default values for these on modern operating systems are typically `C:\ProgramData\ManageSoft Corp` and `C:\Windows\Temp\ManageSoft` (for the SYSTEM account running processes as Windows services).
  - UNIX-like systems: `/var/opt/managesoft` for working data, including `/var/opt/managesoft/log` for log files.

Platform	Package type	Package size	Installed	Workspace
AIX	LPP	25 MB	33 MB	120 MB
HP-UX	SD-UX	74 MB	74 MB	180 MB
Linux i386 (Red Hat, Oracle, CentOS, Fedora, SuSE)	RPM	11 MB	27 MB	120 MB
Linux x86_64 (Red Hat, Oracle, CentOS, Fedora, SuSE)	RPM	36 MB	65 MB	160 MB
macOS	Mac Package Bundle	22 MB	55 MB	150 MB
Solaris SPARC	Sys V Package (pkg)	30 MB	30 MB	120 MB
Solaris x86	Sys V Package (pkg)	25 MB	25 MB	110 MB
Windows	MSI	30 MB	60 MB	160 MB

The following log files are available:

- `installation.log` — Log from `ndlaunch`, responsible for downloading and install all packages required for FlexNet inventory agent
- `policy.log` — Generated by the policy download component, `mgspolicy`
- `schedule.log` — Log from the `ndschedag` schedule component
- `tracker.log` — Generated by the inventory component, `ndtrack`
- `uploader.log` — Log from the `ndupload` file upload component
- `usageagent.log` — Generated by the `mgsusageag` usage tracking service.

Memory requirements

- Minimum RAM: 512 MB
- Recommended RAM: 2 GB

In general, through a cycle of inventory gathering and upload, the memory demand is in the order of 5-30 MB.

Communication protocols and ports

All ports used by FlexNet inventory agent are configurable to any value through preference settings, for example by including the port number in URL settings. The default values for communications supported between adopted devices and inventory beacons are:

- File upload and download using HTTP protocol: port 80
- File upload and download using HTTPS protocol: port 443
- Additional ports may be required if supporting a proxy.

Supported packages to inventory

FlexNet inventory can include data from most package technologies supported by the operating systems, and some additional third-party packaging technologies:

Platform	Supported package technologies
All platforms	InstallAnywhere (IA), InstallShield Multiplatform (ISMP), BEA/Oracle Installer (BEA), Oracle Universal Installer (OUI), IBM Installation Manager (IIM)
AIX	LPP, RPM
HP-UX	Software Distributor SD-UX Package
Linux	RPM (Red Hat, CentOS, Oracle, SuSE, Fedora, etc), DPKG (Debian, Ubuntu).
macOS	Mac Application Bundle, Mac Package Bundle
Solaris	Sys V Package (pkg), IPS
Windows	MSI, Add/Remove Programs Registry Key

However, FlexNet inventory cannot collect data from some of the less common or newer operating system technologies and many third-party technologies. Some known examples include:

- All platforms — IBM InstallStream, IBM Tivoli Netcool Installer
- macOS — Mac flat package.

System load benchmarks

The following notes reflect observed behavior on sample systems using the full installed FlexNet inventory agent.

Task	Run duration (seconds)	CPU usage (seconds)	CPU usage (% of single core)	Memory usage	Network load
Inventory collection	13 to 240 s	5 to 130 s	10% to 50%	4 MB to 20 MB	10 KB to 200 KB per upload
Usage monitoring	Ongoing	Under 1 s / day	Negligible	4 MB to 8 MB	5 KB to 20 KB / day
Policy update*	15 s to 33 s	Under 1 s per policy	Negligible	3 MB to 5 MB	10 KB to 100 KB per policy update

* For each policy update, FlexNet inventory agent downloads its current policy from its preferred inventory beacon. This links to several packages:

- Failover settings (upload and download locations on all available inventory beacons, all of which are configured for anonymous authentication)
- Client configuration (the preferences for use by all installed components included in the FlexNet inventory agent)
- Schedule
- Inventory settings and extensions (`InventorySettings.xml`)
- Agent self-upgrade package.

Using each package, FlexNet inventory agent checks the last downloaded content against the version currently on the inventory beacon. When there is no change, there is no further download (so that the checking process is both rapid and lightweight for the network). Changed items are downloaded, which may increase the network load for that occasion. The largest single impact is when a new agent update package is declared: for details of the download size per platform for update packages, see the disk space listing above.

Adopted: Accounts and Privileges

The complete FlexNet inventory agent deployed automatically through the inventory beacons has distinct security requirements for different phases, and across different platforms.

Platform	Deployment (adoption by inventory beacon)	Operations
Windows	On the target device, a Microsoft service account with local administrator rights to allow for software installation. Optionally, this account may be any of: • A unique account for every target inventory device • An account known in common to a logical group of target devices • A domain administrator account that has installation rights on all target devices. In each case, the account credentials must be saved in the Password Manager on the adopting inventory beacon. (The Password Manager allows for filtering credentials against a group of devices, for example by pattern matching against machine names.) ***Tip:*** *If you choose to use a highly-privileged account, such as a domain administrator, you might also choose to remove it from the Password Manager when all target devices have been adopted. (If you choose this approach, it is best practice when removing the account to disabled any targets that include a setting to adopt target devices, since adoption will fail without an appropriate privileged account.) You may also need to restore the account into the Password Manager to allow for future adoption of newly-added target devices.*	FlexNet inventory agent runs as the local SYSTEM account.

Platform	Deployment (adoption by inventory beacon)	Operations
UNIX-like platforms	An account that allows `sudo` elevation without requiring an interactive password. Installation of the package for FlexNet inventory agent requires `root` level privileges. ***Warning:*** *The user name of the operating system account must not include a hash (#) character, as this causes a failure when attempting to upload the generated `.ndi` files to the application server.*	When the FlexNet inventory agent has been deployed automatically through adoption, it must run as `root` for all its services on the local device. The following security settings apply: • The `/var/opt/managesoft` directory is only accessible by `root`. • The `/opt/managesoft/lib` and `/opt/managesoft/libexec` folders are completely locked down to root only. • The `/opt/managesoft/bin` folder is open to all, to allow easy access to the path of the executables in the folder when using privilege escalation tools like `sudo`. • The executables in the `/opt/managesoft/bin` folder are locked down to root only. • The `/opt/managesoft/documentation` and `/opt/managesoft/software tag` folders are readable by all.

Adopted: Implementation

"Adoption" refers to the process by which FlexNet Manager Suite installs the FlexNet inventory agent on target devices, based on rules that you set.

This approach bypasses the manual preparation of installation packages and their deployment using the third-party deployment tools. In this process, the configuration and deployment of FlexNet inventory agent is fully automated, and you do not need to understand the various code components involved, nor their many settings, nor do any custom configuration.

Tip: *The trade-off is that, while not doing any custom configuration, you also cannot change the default installation location on the target device or choose the agent operation mode on UNIX-like devices. (In contrast, the Agent third-party deployment case allows for custom installation paths for the complete FlexNet inventory agent. The Agent third-party deployment also allows the user to configure whether the installed agent is running with full privilege or least privilege on UNIX-like devices.)*

The main requirements for this Adopted case are targeting the devices through an inventory beacon, and arranging credentials to allow for installation.

To manage automated adoption of discovered devices (summary):

1. Ensure that you have the appropriate inventory beacons fully operational.

   To do this, navigate to **Discovery & Inventory > Beacons** (in the **Network** group), and check the following properties for an existing inventory beacon:

Property	Expected Value
Beacon status	`Operating normally`
Policy status	`Up to date`
Connectivity status	`Connected`

   To deploy and configure a new inventory beacon, click **Deploy a beacon**. Consult the online help for these pages for more information.

2. Ensure that at least one inventory beacon is configured to cover the subnet containing the target inventory devices.

   While all inventory beacons receive all rules declared in the web interface of FlexNet Manager Suite (when they download the `BeaconPolicy.xml` file), each one enacts only those rules that apply to target devices that fall within their assigned subnet(s). This setting is available through the web interface for FlexNet Manager Suite at **Discovery & Inventory > Beacons**. See the online help there for more information.

   Tip: *It is best practice to deploy an inventory beacon into each subnet that contains target inventory devices. This allows the inventory beacon to reliably use ARP or `nbtstat` requests to determine the MAC address of a discovered device (reliability of these results is reduced across separate subnets). Where, across subnets, only an IP address can be found for a device (that is, the device data is missing both a MAC address and a device name), a record is created for the discovered device; but because IP addresses may be dynamic, this is insufficient to allow merging with more complete records (which also contain either or both of the MAC address and a device name). Such complete discovery records may be created automatically when inventory is first returned from the locally-installed FlexNet inventory agent: not finding an existing, complete and matching discovered device record to link with the inventory device record, FlexNet Manager Suite automatically creates one. This means you may see multiple discovered device records with duplicate IP addresses: one record is complete (from inventory), and one or more others are missing identifying data (across subnets) as discussed. These cannot be merged automatically, and you are left with a manual task to clean up incomplete duplicate discovered device records. What's worse, if you have a rule to repeat the discovery process (for example, looking for newly-installed devices) and you still have incomplete discovery data from an inventory beacon reaching across subnet boundaries, the unmatched and incomplete record is recreated at each execution of the discovery rule.*
   In contrast, having a local inventory beacon in the same subnet as target devices provides both the IP address and the MAC address, which is sufficient for matching discovered device records. If you must do discovery across subnet boundaries without a local inventory beacon, ensure that there are full DNS entries visible to the inventory beacon for all devices you intend to discover. This allows the inventory beacon to report both an IP address and a name (either the device name or a fully-qualified domain name [FQDN]), which combination is again sufficient for record matching.

3. If yours is a highly secure, locked down environment, you may need to open network ports on the target computer devices to allow for remote execution.

   Since the inventory beacons use standard ports to access target devices and remotely install the FlexNet inventory agent, the required ports are already available in many environments. (The ports are documented in the online help, under *FlexNet Manager Suite Help* > *Inventory Beacons* > *Inventory Beacon Reference* > *Ports and URLs for Inventory Beacons*. The default requirements for remote execution are ports 445 for SMB on Windows and 22 for SSH on Unix.)

4. Ensure adequate credentials are available for the remote execution process to run. There are two possible approaches for Windows devices:

   - You can register a domain administrator account that has installation privileges on all the target computer devices within the domain. This approach minimizes entries in the Password Manager.
   - You can record appropriate (potentially unique) credentials for each device in the Password Manager. With this approach, you should also add filters to limit the number of password attempts on each target device, so that the remote execution attempt is not terminated because it attempted too many credentials without success.

   These credentials must be recorded in the secure Password Manager available on each inventory beacon (for details, see the online help, under *FlexNet Manager Suite Help* > *Inventory Beacons* > *Password Management Page*).

   For UNIX-like devices, the `ssh` daemon must be installed, and you must either:

   - Record `root` credentials for the target device in the Password Manager on the applicable inventory beacon
   - Record *non*-root credentials for the target device in the Password Manager on the applicable inventory beacon, and additionally ensure that a tool to allow privilege escalation (such as `sudo` or `priv`) is installed on target devices and either:
     a. The use of that tool is configured in the Password Manager (in the extra fields exposed when you specify and SSH account type), or
     b. Target devices are configured to allow escalation of privileges without requiring an interactive password.

5. For gathering Oracle inventory from UNIX-like platforms, ensure that you have authorized FlexNet inventory agent version 13.2.0 or later for adoption and self-managed upgrades.

   Earlier versions of FlexNet inventory agent may fail to collect Oracle inventory on UNIX-like systems where permissions prevent global access to Oracle directories or files. For details about setting the permitted version of FlexNet inventory agent, see Adopted: Specifying an Installed Agent Upgrade.

6. Set the schedule for operations of the FlexNet inventory agent in the web interface of FlexNet Manager Suite (navigate to **Discovery & Inventory > Settings**, in the **Inventory agent schedule** section).

7. Navigate to **Discovery & Inventory > Discovery and Inventory Rules**, and create one or more rules to take inventory from target computing devices within your enterprise, and then at the beginning of the inventory process, adoption occurs when in policy for the target device.

   Rules consist of

   - **Targets** that identify sets of devices, and (for all the devices identified within a single target) specify policy about how to connect, whether to collect CAL evidence, whether to track application usage, and whether to adopt — all devices intended for adoption must be included in at least one target that has **Allow these targets**

to be adopted selected (and at the same time, must *not* be included in any overlapping target for which **Do not allow these targets to be adopted** is selected, as a 'deny' always over-rides an 'allow').

Tip: *In the case of these policy settings such as adoption, targets need not be used in rules to have effect. Policy is determined by the net effect of the policy settings on all the targets that apply to a given device. This policy setting is a separate function of targets, independent of their possible use in rules. Secondly and separately, targets are also used to identify which devices a rule should act on. If a device is covered in at least one target used in a rule with an inventory gathering action (as listed next — this conditions ensures that an inventory beacon touches the target device), its adoption policy (allow or deny) can be specified in any overlapping target (overlapping because it covers the same device), regardless of whether the overlapping target is actually used in a rule.*

- **Actions** that declare what to do to the targeted devices — to ensure adoption, the relevant rule must include at least one of the following inventory settings when you create or edit an action (the action may also include discovery settings, or alternatively you may look in the **Discovery of devices** area and select **Use previously discovered devices**):
  - In the **General devices discovery and inventory** section (click the title bar to expand the section), **Gather hardware and software inventory from all target devices** selected
  - In the **Microsoft Hyper-V discovery and inventory** section, both the **Discover Microsoft Hyper-V** check box and the **Gather Microsoft Hyper-V hardware and software inventory** check box selected; *and* Hyper-V is then discovered on the device
  - In the **Microsoft SQL Server discovery and inventory** section, both the **Discover Microsoft SQL Server** check box and the **Gather Microsoft SQL Server hardware and software inventory** check box selected; *and* SQL Server is then discovered on the device.
- A schedule for implementing the action on the targeted devices.

Tip: *Part of the art in this automated deployment may be in declaring a schedule that suits when the target devices are available (that is, running, connected to the network, and not too busy). Particularly with individual workstations and laptops, this may require some external process management. For example, you may communicate to users that they should leave target devices running overnight, or some similar arrangement. Since the discovery and adoption rules execute on a repeated schedule, there are many such 'windows' when devices can be automatically adopted.*

By specifying multiple targets, you can choose which computer devices are adopted. For more information, see the online help for these pages.

8. Wait for the execution of the rule(s), the installation of the FlexNet inventory agent, and the resultant data uploads.

   When the inventory beacon contacts a target device listed for gathering inventory, it checks the operating system and checks whether the full FlexNet inventory agent is already installed. On a target device that is listed in policy for adoption, *but* where the FlexNet inventory agent is not already present, the inventory beacon automatically installs the FlexNet inventory agent (that is, 'adopts' the device). No methods of inventory gathering other than by the locally-installed FlexNet inventory agent are ever used on target devices for which the policy (net of all targets) is adoption.

   To monitor progress and results, navigate to the system menu (▼ in the top right corner), **System Health >**

System Tasks, and see the online help there for more information. You can also navigate to **Discovery & Inventory > Discovery and Inventory Rules**, and select the **Rules** tab. On that page, click the name of any rule to expose additional details, including a table of **Adoption results**.

Tip: *It is important to check for successful adoption. Should it happen for any reason that adoption fails, the fact that adoption has been set for the target device prevents inventory collection through (for example) the Zero-footprint process of inventory gathering. This combination may mean that no inventory is returned for the device at all.*

When initial execution of the rules is successfully completed:

- On the adopted inventory device, the FlexNet inventory agent is by default installed:
  - On Windows, in `C:\Program Files (x86)\ManageSoft`
  - On UNIX-like platforms, in `/opt/managesoft`.
- The adopted devices are visible in the web interface of FlexNet Manager Suite. For example, they are listed in **Discovery & Inventory > All Discovered Devices** page.After an inventory upload and import, the **Agent installed** column shows that the FlexNet inventory agent has been successfully deployed and is reporting.
- **Discovery & Inventory > Settings > Inventory agent schedule** section, your adopted devices start reporting inventory collected by the FlexNet inventory agent. After subsequent inventory imports and compliance calculations, the results are visible in the management and reporting views within FlexNet Manager Suite.
- The targets you declared to set policy for adoption now have no further effect on adopted devices (they do not, for example, unnecessarily cause repeated installations). Each installation of FlexNet inventory agent is a state-based machine controlled by its policy, prepared for it on demand by an inventory beacon. However, keeping the policy-setting targets in place is best practice, for at least two reasons:
  - If you remove a target that specifies a policy combination (adoption, connection, CAL evidence and usage tracking), you may inadvertently switch behaviors of the installed agents
  - The same policy setting in favor of adoption also prevents overlapping gathering of FlexNet inventory by other means, such as the Zero-footprint case.
- Any changes to policy settings affecting installed FlexNet inventory agents are transmitted to all inventory beacons, and automatically included in subsequent policy updates on the next policy request by each installed FlexNet inventory agent.
- Through policy, you can also control the self-update mechanism when new versions of the FlexNet inventory agent are ready to deploy (for details, see Adopted: Specifying an Installed Agent Upgrade).

Note: *For UNIX-like platforms (only), you can manually update the preferences controlling the behavior of the installed FlexNet inventory agent. From a console or remote terminal connection, run the command:*

```
/opt/managesoft/bin/managesoft-configure
```

This will prompt for configuration items.

Adopted: Specifying an Installed Agent Upgrade

If the FlexNet inventory agent has been deployed automatically through inventory beacons, it manages its own self-updates to align with the version currently specified as part of the policy downloaded to all managed devices. Use the procedure described in this topic to define which version is authorized for use on specified platforms.

Even when there is an update to the central application server, the upgrade mechanism for FlexNet inventory agents is turned off. This permits operators with Administrator level rights the freedom to manage the upgrade of deployed FlexNet inventory agents independently of upgrades to other parts of the system.

Keep in mind that the self-update mechanism works either way:

- In the most common case, if you specify a *later* version for FlexNet inventory agent than the one currently running on a target inventory device, you are specifying an upgrade.
- If you specify an *earlier* version for FlexNet inventory agent than the one currently running on a target inventory device, you are specifying a downgrade. This mechanism can be used to roll back a version of FlexNet inventory agent, if this ever becomes necessary.

Tip: *Instances of FlexNet inventory agent installed on platforms that you do not select are unchanged through policy, and remain at their current installed versions, neither upgraded nor downgraded.*

Because non-selected platforms are completely unaffected by the current setting, you can use (and re-use) these controls to work through scenarios such as:

- Release version *X* to Windows devices when the team administering Windows has completed their testing. Weeks later perhaps, when the Linux team finishes *their* testing, a new setting on this page authorizes version *X* for the Linux platform, while other platforms are unaffected.
- You roll out a new version across all platforms, but then find an issue affecting only Solaris platforms. A setting that selects only the two Solaris platforms, and chooses an earlier, known good version of FlexNet inventory agent, automatically rolls back the FlexNet inventory agent on all devices running Solaris. Later, when a new version is available that addresses the issue, you can authorize that repaired version for the Solaris platforms, as well as for any other platforms you wish.

You may change these settings frequently to manage scenarios like the above; but in general, you should leave an interval of at least 48 hours between changes, and longer if you have devices with intermittent connectivity (such as notebooks carried by road warriors). This allows time for each specification to complete this roll-out process:

1. Each inventory beacon, by default, checks every 15 minutes for policy updates and any related installation packages. If you have a hierarchy of inventory beacons, new files must trickle down from one to the next.
2. Each installation of FlexNet inventory agent checks at a random time once every 12 hours for any policy updates. Of course, mobile devices that are away from the office may need to wait for a return to full connectivity.
3. On affected platforms, FlexNet inventory agent must download any necessary installation packages, self-update, and resume operations.
4. On your global schedule, each FlexNet inventory agent collects and uploads inventory from connected devices.
5. Overnight (by default), there is a full inventory import and compliance calculation.
6. Thereafter you can check deployed versions in the **Discovery & Inventory > FlexNet Inventory Agent Status** page.

Tip: *This inventory setting grants permission (through policy) to the FlexNet inventory agents to perform self-upgrades or downgrades to the specified version. The setting, therefore, can only be put into effect on those platforms where the FlexNet inventory agent includes self-update functionality. Currently, FlexNet inventory agents on Debian or Ubuntu Linux do not include self-update functionality. On these platforms, you can do any of:*

- *Deploy new versions of FlexNet inventory agent manually*
- *Use your preferred third-party deployment tool to publish updates to FlexNet inventory agents*
- *Uninstall the old version(s) of FlexNet inventory agent, and once again target the devices for adoption through FlexNet Manager Suite.*

For supported platforms, the available controls in the web interface let you choose a method of management ranging from preventing self-updates entirely, through specifying the version to run on each platform type, to allowing continuous updates to the latest version.

Tip: *If you are adopting a UNIX-like server for Oracle inventory gathering by the installed FlexNet inventory agent, be sure to set the version of FlexNet inventory agent to 13.0.1 or later for this platform type. Earlier releases of the FlexNet inventory agent may fail to collect Oracle inventory on servers where permissions prevent global access to Oracle directories or files.*

To specify the required version of FlexNet inventory agent on selected platforms:

1. In the web interface for FlexNet Manager Suite, navigate to **Discovery & Inventory > Settings**.

   The **Inventory Settings** page is displayed, provided that your operator account is in the `administrator` group.

2. Scroll down to the **Inventory agent for automatic deployment** section.
3. For **Upgrade mode**, choose one of the following:
   - `Do not upgrade automatically` — There are no automated self-upgrades of FlexNet inventory agent on any platform. You take responsibility for upgrades using separate technologies or methods. With this choice, you may click **Save** immediately.
   - `Upgrade selected platforms to specific version` — With this choice, additional controls are revealed. Continue with the following steps.
4. For **Version to deploy**, set the target version of FlexNet inventory agent that should be used for upgrades on the platforms you will select next:
   - *nn.nn.nn* `(latest)` — The most recent release available on the application server is marked with `(latest)` in brackets. Only on the platforms you select in the next control, the FlexNet inventory agent self-upgrades to the latest version once this has been downloaded to an accessible inventory beacon.
   - *nn.nn.nn* — All upgrade packages currently available on the application server are listed. You may select any version, later *or earlier* than one currently deployed on target inventory devices running the operating systems you target in **Platform options**.
5. For **Platform options**, select (check) at least one, or several, or all of the available platforms.
6. When you are satisfied with your settings, click **Save**.

Your changes are saved in the compliance database, overwriting the previous settings. The roll-out of your new

specification starts, as described above. This new policy setting is directed only to your selected platforms, leaving other platforms with their previous settings saved locally on the inventory devices, but no longer visible in the web interface.

Adopted: Troubleshooting Inventory

Inventory gathering and upload is a sophisticated chain from target inventory device through inventory beacon to central application server. For general trouble-shooting over the whole process, see the online help for FlexNet Manager Suite under *Inventory Beacons* > *Inventory Beacon Reference* > *Troubleshooting: Inventory Not Uploading*. This topic focuses entirely on inventory collection on the target inventory device.

The regular log file for the `ndtrack` executable is identified in `[Registry]\ManageSoft\Tracker\CurrentVersion\LogFile` (see LogFile (inventory component)). In the Adopted case, the default paths are:

- On Windows platforms, `$(TempDirectory)\ManageSoft\tracker.log`
- On UNIX-like platforms, `/var/opt/managesoft/log/tracker.log` (when the `ndtrack` executable runs as root).

For advanced trouble-shooting, you may require more advanced tracing and logging. You may also be asked to submit a trace file to assist the Support team at Flexera to solve difficult problems in your environment.

To configure advanced tracing for the installed FlexNet inventory agent:

1. In a flat text editor, open the `etcp.trace` file.

   In the Adopted case, this file is co-located with the installed `ndtrack` executable on the target inventory device:

   - On Windows, the default is `C:\Program Files (x86)\ManageSoft\etcp.trace`
   - On UNIX-like platforms, the default is `/opt/managesoft/etcp.trace`.

2. Configure the name and location of the trace/log file that will be generated on the inventory device.

   The hash or pound character (#) identifies a comment. To "uncomment" a line in the `.trace` configuration file means to delete (only) the leading hash character. Choose one of the following lines, uncomment it, and optionally modify it to your requirements. On Windows:

   ```
   #filename=C:\ManageSoft.log
   #filename=C:\ManageSoft%p_%d_%t_%u.log        # filename pattern with everything!
   ```

   On UNIX-like platforms:

   ```
   #filename=/tmp/log/mgstrace.log
   #filename=/tmp/log/ManageSoft%p_%d_%t_%u.log # filename pattern with everything!
   ```

   See the notes within the file header for the use of the supported variables within the file name.

   Tip: *It is best practice to use a pattern that includes (at least) either a date stamp (%d) or a sequential number (%u). Without these, the fixed file name means tracing information is appended to the same file with every inventory collection. This can quickly produce a trace file too large for text editors to read, and too hard to manage in terms of disk space. Variables in the file name trigger creation of a new file each time the associated variable is changed (or, for %u, at every invocation of `ndtrack`).*

Important: *The log file path:*

- *Must be on the same drive as the* `ndtrack` *executable (on Windows devices)*
- *Must exist and be writable before the* `ndtrack` *executable is next invoked (tracing does not create any directories, and does not function if any directory in the specified path is missing or unwritable).*

3. Uncomment the lines for which you want to enable tracing (ensuring that the uncommented line now starts with a plus sign).

   The tracing controls are arranged hierarchically. For example, uncommenting the one line for `+Inventory/Tracker` enables tracing for all child controls, as in this example:

```
+Inventory/Tracker
#+Inventory/Tracker/Preferences
#+Inventory/Tracker/Environment
#+Inventory/Tracker/Hardware
#+Inventory/Tracker/Hardware/WMI
#+Inventory/Tracker/Hardware/WMI/Class
#+Inventory/Tracker/Hardware/WMI/Instance
#+Inventory/Tracker/Hardware/WMI/Property
#+Inventory/Tracker/Hardware/Processor
#+Inventory/Tracker/Hardware/Memory
#+Inventory/Tracker/Hardware/Hypervisor
#+Inventory/Tracker/Hardware/DiskDrive
#+Inventory/Tracker/Registry
#+Inventory/Tracker/Registry/Keys
#+Inventory/Tracker/Registry/Values
#+Inventory/Tracker/Package
#+Inventory/Tracker/Package/Info
#+Inventory/Tracker/Software
#+Inventory/Tracker/Software/Directory
#+Inventory/Tracker/Software/File
#+Inventory/Tracker/Software/Version
#+Inventory/Tracker/Oracle
#+Inventory/Tracker/Oracle/Listener
#+Inventory/Tracker/Generate
#+Inventory/Tracker/Compress
#+Inventory/Tracker/Upload
```

   This setting enables (almost) *all* tracing related to the tracker component (`ndtrack`). You can also create an exemption to the general setting by making the appropriate line starts with a minus sign. For example, this one-line change within the above:

```
-Inventory/Tracker/Registry
#+Inventory/Tracker/Registry/Keys
#+Inventory/Tracker/Registry/Values
```

   turns off tracing for everything related to gathering inventory from the Windows registry (that is, the disable setting is also inherited by the `Inventory/Tracker/Registry/Keys` and `/Values` controls).
   One control that may affect the tracker component is outside this set. Because the tracker attempts an upload to

the inventory beacon as soon as inventory gathering is complete, its tracing is affected by the `+Communication/Network` setting (along with the `ndupload` and `ndlaunch` components). This enables tracing of all network communications, including server certificate checking and the like.

Some common choices for tracing the inventory gathering process are listed in the table below.

4. To turn off tracing for an individual line that has previously been enabled, either comment out the line again, or switch the plus sign to a minus sign at the start of the line. A quick way to turn off tracing but keep all the settings for future use is to comment out only the `filename` setting that specifies the log file.

Some of the more commonly used tracing options for the tracker include the following:

Category	Option	Notes
Networking	`+Communication/Network`	Traces all low-level upload and download actions (whether HTTP or HTTPS). It includes HTTPS certificate checking and related areas. Covers actions by the `ndtrack`, `ndupload`, and `ndlaunch` components.
All inventory	`+Inventory`	Traces all inventory operations, which on large inventory tasks, could result in a sizable trace file.
All tracker	`+Inventory/Tracker`	This traces almost all operations of the `ndtrack` executable.
Preferences	`+Inventory/Tracker/Environment`	Shows active preferences, whether set in the registry (on Windows platforms) or in the `config.ini` file (on UNIX-like platforms), or inbuilt default values.
Hardware inventory	`+Inventory/Tracker/Hardware`	Traces all hardware inventory classes visible in the `<Hardware>` node in the `.ndi` inventory file, including CPU information and virtualization.
Software inventory	`+Inventory/Tracker/Package`	Traces the inventory operations that populate the `<Package>` nodes in the `.ndi` inventory file.
Software inventory	`+Inventory/Tracker/Software`	Tracing mainly for file inventory gathering (such as the `<Content>` and `MD5` nodes of the `.ndi` file).
Oracle inventory	`+Inventory/Tracker/Oracle`	When the inventory device hosts Oracle Database, this is the tracing for local Oracle inventory.
Operations	`+Inventory/Tracker/Generate`	Traces the preparation of the `.ndi` inventory file(s), keeping in mind that on an Oracle Database server, there may be multiple files generated.
Operations	`+Inventory/Tracker/Compress`	Traces the compression of the `.ndi` file into a `.gz` archive.

Category	Option	Notes
Operations	`+Inventory/Tracker/Upload`	Traces the upload of the `.ndi.gz` archive to an inventory beacon by the tracker. ***Tip:*** *If the immediate upload by the tracker fails for some temporary reason, the upload is attempted again later by the `nduploa`d component. While this component does not provide this same level of operations tracing as the tracker does, you can enable `+Communication/Network` for low-level tracing of each step in the upload interaction.*

3

Agent Third-Party Deployment: Details

This chapter provides great detail about the FlexNet inventory agent deployed by methods of your own choosing (*not* using FlexNet Manager Suite to perform adoption), and installed on target devices for subsequent control by downloaded policy (labeled the Agent third-party deployment case in Deployment Overview: Where to, and How). Keep in mind that this label is a shorthand for "the full FlexNet inventory agent deployed using third party tools". If you are looking for details about how in general to deploy inventory-gathering code elements, see the *Implementation* topic within the *Details* chapter for each of the methodological cases.

This document provides a consistent set of data (as far as possible) across all the different use cases, each in its own chapter. This means that, once you have chosen your preferred use case, you can focus only on the details for that one, and ignore all other use case chapters.

Because FlexNet inventory collection supports a range of operating system platforms, some topics necessarily contain several sets of information, and you can select only those details that apply to your chosen platform(s).

In addition to the distinct chapters for the different use cases, you should also review the subsequent chapter on functionality that is common throughout. This is followed by detailed reference material on command lines, preferences, file formats, and the like.

Agent Third-Party Deployment: Normal Operation

This topic assumes that deployment and installation is complete (for those details, see Agent Third-Party Deployment: Implementation and its subtopics), and describes operations thereafter (to help you decide whether to proceed with this particular case). It describes the operation of FlexNet inventory agent once you have used your preferred third-party tool to deploy it into locations within your computer estate where each installation can access one or more inventory beacons, and function normally (labeled the Agent third-party deployment case for short). These computer devices are to become "inventory devices" within FlexNet Manager Suite.

The operating procedures for the full FlexNet inventory agent deployed in this way are consistent across Microsoft Windows and UNIX-like systems, with some obvious distinctions in system dependencies like file paths. The entire process is covered, including what happens to the collected data after it is uploaded by the FlexNet inventory agent.

Each numbered step provides a summary point, followed by further specific details that you can skip over until needed.

1. Normally only once (probably even before deploying FlexNet inventory agent, although you can certainly change it later if required), you set the schedule for operations of the FlexNet inventory agent in the web interface of FlexNet Manager Suite (navigate to **Discovery & Inventory > Settings**, in the **Inventory agent schedule** section).

   This schedule is automatically downloaded to all inventory beacons for delivery to installed FlexNet inventory agents on your inventory devices.

   Tip: *Inventory operations of the installed FlexNet inventory agent in the Agent third-party deployment case are never controlled by inventory rules created in the web interface. This includes the schedules that form part of rules, which have no effect on inventory gathering by the installed FlexNet inventory agent (which is a state-based device, self-managing to align with its policy). It is for this reason that the schedule for on-going inventory operations in the Agent third-party deployment case are set as described above, rather than as part of discovery and inventory rules.*

2. Components of the FlexNet inventory agent are triggered by a long-running process (`ndinit` on Windows, and `ndtask` on UNIX-like platforms). The process is restarted automatically after a machine reboot.

   For more information about the services on UNIX-like platforms, see Agent Third-Party Deployment: Services on UNIX. For full architectural details of the FlexNet inventory agent, see Agent Architecture.

3. Immediately upon first installation, each FlexNet inventory agent asks its bootstrap inventory beacon for its initial policy. Thereafter at a random time once every 12 hours, it checks for policy updates.

   The bootstrap inventory beacon is the one you identified in the bootstrap configuration file (`mgssetup.ini` for Windows devices or `mgsft_rollout_response` for UNIX-like systems). Once the initial policy is successfully downloaded, the installed FlexNet inventory agent has a complete list of all existing inventory beacons, and makes future policy update requests to the most appropriate one (by default, this is one in the same Active Directory site with the fastest ping response time at each upload time; but if these conditions cannot be met, it may be a random choice). Any inventory beacon will respond to a query from any installed FlexNet inventory agent (and calculate and provide a policy file for it), provided that the system is not in migration mode. Migration mode is set in the web interface for FlexNet Manager Suite (navigate to **Discovery & Inventory > Settings > Beacon settings > Migration mode: Restrict inventory settings to targeted devices**).

   (For details of the policy file, see Policy Files (.npl).) The FlexNet inventory agent downloads and checks the package files that are linked in the downloaded policy. The package files are very small and quick to download and process, and each identifies the version of its related content file. If the FlexNet inventory agent determines that no content files have changed since they were last collected, no further downloads take place at this time. If anything has changed, the changed content is downloaded and the appropriate settings on the inventory device are updated. These potential downloads include any change to the operational schedule for inventory collection.

4. If usage tracking is in policy for this device, the usage component monitors the running processes on the system, recording the number of times, and for how long, each process is run.

   This component does not have any noticeable impact on system performance. (Each application being tracked adds about 250 bytes of compressed data to the upload packages.) It looks up the OS-standard installation data (such as MSI on Windows, RPM on Linux, and so on) to associate a process with the installation evidence and file paths. Technically, the usage component `mgsusageag` is a daemon on UNIX-like systems, and on Windows it's a plug-in to `mgssecsvc.exe`, which starts as a Windows service at system start-up (for further details, refer back to Agent Architecture). After the results are uploaded, usage can be reported only against applications for which there is an independent installation record. (Usage results are not used to create an installation record, since the

application may have been removed within the time window where usage is tracked.)

5. Apart from usage tracking, the FlexNet inventory agent sits dormant on the target inventory device until the scheduled time for inventory collection.

   The scheduled to-do list is managed by the `ndtask` component (this is a cross-platform component matching the functionality of `mstask`). When a schedule trigger fires, `ndtask` runs `ndschedag` (passing it a GUID for the required action), and for inventory collection `ndschedag` runs the `ndtrack` component. Notice that `ndschedag` provides its own logging in `scheduler.log`, in the following default directories (there are several `LogFile` preferences that can override these default locations):

Windows platforms	`$(TempDirectory)\ManageSoft\`
UNIX-like platforms	`/var/opt/managesoft/log`

6. The `ndtrack` executable by default runs at low priority to collect software and hardware inventory details on the local device.

   This means that higher priority tasks are not interrupted, so that there is minimal impact on system performance. Inventory details are saved in an `.ndi` file on the local file system on the inventory device, by default:

Windows platforms	`$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Tracker\Inventories`
UNIX-like platforms	`/var/opt/managesoft/tracker/inventories`

   This location may be altered with the `MachineInventoryDirectory` preference. This directory preserves the local copy of the inventory file (where you can inspect its contents) until it is over-written at the next inventory collection by the same account (since inventory file naming reflects the account running the inventory collection).

   At the same time, a compressed (`.ndi.gz`) copy of the file is also saved, ready for upload, in a separate directory:

Windows platforms	`$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Common\Uploads\Inventories`
UNIX-like platforms	`/var/opt/managesoft/uploads/Inventories`

   If the FlexNet inventory agent has uncovered any Oracle services running on the local device (and, for UNIX-like target devices that have the FlexNet inventory agent running in the full privilege default operation mode, only when the FlexNet inventory agent is running as `root`), a second `.ndi.gz` file of Oracle inventory is also generated, and saved in the same directory (an uncompressed version is not saved). As well, the Oracle discovery is reported in an uncompressed `.disco` file, saved in the `Discovery` directory (a peer of the `Inventories` directory in the uploads set). Since the behavior of the installed FlexNet inventory agent is not controlled by inventory rules set in the web interface, this Oracle discovery and inventory does not rely on those rules, and will occur even when no rules for Oracle inventory collection exist, provided that:

   - `InventorySettings.xml` is available to the FlexNet inventory agent (in the folder identified in the `InventorySettingsPath` preference setting, which defaults on Windows to `$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Tracker\InventorySettings\` and on UNIX-like platforms to `/var/opt/managesoft/tracker/inventorysettings` — if either the preference or

the file is missing, Oracle inventory is skipped)

- For UNIX-like target devices, the FlexNet inventory agent is running as `root` (for the installed FlexNet inventory agent, all Oracle discovery and database inventory gathering are blocked for any non-`root` account).

Tip: *A system trace on the UNIX version of `ndtrack` shows that it reads `/etc/passwd`. The UNIX `ndtrack` uses the `setpwent()` and `getpwent()` library calls to obtain the `pw_name`, `pw_dir` and `pw_shell` properties for each user. In particular, `ntrack` uses the `pw_dir` property (each user's home directory) to find user-based installation evidence for BEA and InstallAnywhere installation technologies.*

7. After collecting the hardware and software inventory, `ndtrack` immediately attempts to transfer the compressed inventory file(s) to an inventory beacon.

   The upload is a background process that does not take priority away from other current tasks running on the inventory device. The destination for the upload is `ManageSoftRL` (a web service on the inventory beacon), which on free-standing inventory beacons saves the `.ndi.gz` file(s) to the folder *`%CommonAppData%`*`\Flexera Software\Incoming\Inventories` on the inventory beacon.

   Tip: *If the inventory beacon is co-located on your central application server (or in large-scale systems, the batch server), the `ManageSoftRL` web service does not save to disk, but instead saves directly to the database as described in step 9.*

   (The upload functionality is shared between the `ndtrack` and `ndupload` components. Because the upload here is attempted as part of inventory collection, upload logging for this event is in the `ndtrack` or `tracker` logs, in the path given in step .)

   - If the initial upload is unsuccessful for any reason, there is a catch-up task on the inventory device that triggers a retry of the upload. (If there are no files awaiting upload at catch-up time, this process shuts down immediately.)
     - *Timing:* The catchup schedule is internal (downloaded to the FlexNet inventory agent in the `.nds` schedule referenced in policy), and cannot be modified in any user interface. The task is called `Upload Client Files` and occurs once daily within a one hour window, the start time for which is randomized by the inventory beacon when it regenerates the schedule file after each update of beacon policy (that is, after each change to rules or schedules in the web interface for FlexNet Manager Suite). If the inventory device is turned off at the time, there is also a catch-up on machine restart.
     - *Logging:* For this catch-up, the upload uses the `ndupload` component, so that logging for the catchup attempt is in the `ndupload` logs.
   - Once the upload is successful (either originally or in catch-up), the copy of each compressed inventory file in the `...\Uploads\Inventories` folder on the inventory device is deleted (meaning that the absence of files after the upload is a sign of success, and the continued presence of files after upload attempts is a sign of failure, with stale `.ndi.gz` files overwritten at the next inventory collection). After success, the process continues below.

8. FlexNet Beacon (the code entity on the inventory beacon) uploads the inventory data to its parent on a schedule set by the Microsoft Scheduled Task `Upload FlexNet logs and inventories` (by default, repeating every minute throughout the day).

The checking cycle when the folder is empty is very quick and does not perceptibly load the inventory beacon, even though it is frequently repeated. The parent of an inventory beacon may be the central application server, or another inventory beacon if these have been arranged in a hierarchy. In the latter case, each inventory beacon in turn repeats the upload process until the data reaches the application server.

9. On the application server (or, in a scaled-up system with separate servers, the inventory server), the web service `ManageSoftRL` receives the uploaded packages for both inventory and (if configured) usage tracking (and other uploaded files).

   These are processed immediately, being loaded into the internal operations databases: inventory (`.ndi`) and usage (`.mmi`) files are loaded into the inventory database; any Oracle discovery (`.disco`) file is loaded into the compliance database. If the service gets overloaded, it will temporarily spool incoming files to its local *%CommonAppData%*`\Flexera Software\Incoming\Inventories` directory (or the peer `Discovery` folder for any Oracle discovery file). From these folders, file import is resumed under the control of Microsoft scheduled tasks (for example, `Import inventories`, which is triggered every 10 minutes).

10. On the next inventory import and license consumption calculation, the inventory and usage data is collected from the inventory database, socialized as necessary, and imported into the compliance database. Here it is used in license calculations, and made available in management views and reports.

    This import step can be triggered in one of three ways:

    - Normally, the batch scheduler triggers an import daily (by default, at 2am local time on your application server), with the license consumption calculation triggered thereafter. This default time is configurable by editing the Microsoft scheduled task `Inventory import and license reconcile` on your application server (or, in larger implementations, batch server).
    - An operator in the `Administrator` role can choose to import the waiting inventory and trigger license consumption calculation, or reconciliation, as soon as possible (navigate to **License Compliance > Reconcile**).
    - For testing, a knowledgeable system administrator could use a command line on your application server (or, in a scaled-up system, your batch server) like:

      ```
      BatchProcessTask.exe run InventoryImport
      ```

      (for details, see the *Server Scheduling* chapter in *FlexNet Manager Suite System Reference*).

Agent Third-Party Deployment: Services on UNIX

The installers for FlexNet inventory agent on UNIX-like platforms install two services which are automatically started after a successful installation and on every system boot. These services are:

- `ndtask` — The agent's task launcher. This runs `ndschedag` which executes further command lines (for example, for `ndtrack`, `ndupload`, `ndlaunch`) to perform the required actions.
- `mgsusageag` — The usage tracking agent.

The services are managed using the operating-system-specific service management tools. For example, to start `ndtask`:

Platform	Command
AIX (see note)	`/usr/bin/startsrc -s ndtask`
HP-UX	`/sbin/init.d/ndtask start`
Linux	`/etc/init.d/ndtask start`
Mac OS X	`/sbin/launchctl start com.flexerasoftware.ndtask`
Solaris	`/etc/init.d/ndtask start`

Similarly, to stop the `ndtask` service:

Platform	Command
AIX	`/usr/bin/stopsrc -s ndtask`
HP-UX	`/sbin/init.d/ndtask stop`
Linux	`/etc/init.d/ndtask stop`
Mac OS X	`/sbin/launchctl stop com.flexerasoftware.ndtask`
Solaris	`/etc/init.d/ndtask stop`

The `mgsusageag` service can be started and stopped in exactly the same way.

Note: *For AIX, the agents are in a group called `managesoft` and can both be started using `startsrc -g managesoft` and stopped using `stopsrc -g managesoft`.*

Agent Third-Party Deployment: System Requirements

The following details apply to the full FlexNet inventory agent when deployed to a target device using third-party deployment tools/methods of your own choice.

Supported platforms

The FlexNet inventory agent operates on the following platforms (inventory targets):

Microsoft Windows

- Windows Server 2008 R2 x64, 2012, 2012 R2, 2012 R2 SP1, 2016, 2019, 2022
- Windows Server Core 2008, 2008 R2 x64, 2012, 2012 R2
- Windows Server Standard (previously known as Windows Server Core) 2016, 2019
- Windows Vista, 7, 8, 10, 11

Note: *Windows systems that can run on ARM-based devices are also supported.*

UNIX-like platforms

- AIX 7.1 LPARs with Technology Level 5 or later, AIX 7.2
- Amazon Linux 2, 2023 (ARM64/AArch64; x86, 32-bit and 64-bit)
- CentOS 6-7.5 (x86, 32-bit and 64-bit); 7.6-9 (x86 64-bit only)
- Debian Linux 8–11.3 (x86, 32-bit and 64-bit); 11.5, 12.0 (x86 64-bit only)

Note: *For Debian Linux 9.4, 9.5 and 10 (both 32-bit and 64-bit), minimal installations of the OS core omit the `ifconfig` command. This prevents collection of the IP address and MAC address in inventory. The `root` user can use the following command to restore this functionality:*

```
apt-get install net-tools  -y
```

- Fedora Linux 26 (x86, 32-bit and 64-bit); 27-38 (x86 64-bit only)
- HP-UX 11i v3 (Itanium only), vPars/nPars
- macOS 10.15.4–14 (applicable for both Intel and Apple M-series processors)

Note: *If you need to run the FlexNet inventory agent of version 20.1 or earlier on an Apple M-series processor ("Apple silicon"), Rosetta 2 must be installed and running. This is Apple's solution for transitioning most Intel-based applications to run on Apple silicon. There are two possible command formats for installing Rosetta 2:*

- *Interactive installation that asks for agreement to the Rosetta 2 license:*

```
/usr/sbin/softwareupdate
    --install-rosetta
```

- *Non-interactive installation:*

```
/usr/sbin/softwareupdate
    --install-rosetta
--agree-to-license
```

Microsoft Windows	UNIX-like platforms
	• Nutanix AHV hypervisor ***Note:*** *All versions that are within their product support lifecycle and use the* `libvirt` *library are supported.* • OpenSUSE Leap 42.2, 42.3 (x86, 32-bit and 64-bit); 15-15.4 (x86 64-bit only) • Oracle Linux 6.0–6.10 (x86, 32-bit and 64-bit); 7.0-8.7 and 9.0-9.1 (x86 64-bit only) • Photon OS 3.0-5.0 • Red Hat Enterprise Linux (RHEL) 6.0-6.10 (x86, 32-bit and 64-bit); 7.0-8.8 and 9.0-9.2 (x86 64-bit only) • Rocky Linux 8, 9 • Solaris 10–11 (SPARC), Zones for versions 10–11 • Solaris 10–11.4 (x86), Zones for versions 10–11 • SUSE Linux Enterprise Server 12 SP3, 12 SP4, 12 SP5, 15, 15 SP1, 15.2-15.5 (x86 64-bit only) • Ubuntu 14–17.04 (x86, 32-bit and 64-bit); 17.10-23.04 (x86 64-bit only) Linux on IBM zSystems platforms are also supported. The following Linux distributions are certified for usage on the IBM zSystems architecture. For details about supported versions on different hardware types, see https://www.ibm.com/support/pages/linux-ibm-z-tested-platforms. • Red Hat Enterprise Linux • SUSE Linux Enterprise Server • Ubuntu

Disk space requirements

The following table gives three disk space figures for installation of FlexNet inventory agent:

- **Package type** gives the kind of installer package provided for each platform, and **Package size** defines the disk space to download or copy the installer for each of the platforms, before installation.
- **Installed** defines the disk space for the binary files after installation. The default locations for installation are:
  - Windows: `%ProgramFiles(x86)%\ManageSoft` (on modern 64-bit systems, this expands to `C:\Program`

Files (x86)\ManageSoft)

- UNIX-like systems: /opt/managesoft

- **Workspace** approximates a typical operating space requirement. The precise requirements depends on the self-update installer package size, the number of inventory files awaiting upload, usage tracking, the growth of log files, and the like. The default locations for this requirement are:
  - Windows: %ProgramData%\ManageSoft Corp for data; and %temp%\ManageSoft for log files. The default values for these on modern operating systems are typically C:\ProgramData\ManageSoft Corp and C:\Windows\Temp\ManageSoft (for the SYSTEM account running processes as Windows services).
  - UNIX-like systems: /var/opt/managesoft for working data, including /var/opt/managesoft/log for log files.

Platform	Package type	Package size	Installed	Workspace
AIX	LPP	25 MB	33 MB	120 MB
HP-UX	SD-UX	74 MB	74 MB	180 MB
Linux i386 (Red Hat, Oracle, CentOS, Fedora, SuSE)	RPM	11 MB	27 MB	120 MB
Linux x86_64 (Red Hat, Oracle, CentOS, Fedora, SuSE)	RPM	36 MB	65 MB	160 MB
macOS	Mac Package Bundle	22 MB	55 MB	150 MB
Solaris SPARC	Sys V Package (pkg)	30 MB	30 MB	120 MB
Solaris x86	Sys V Package (pkg)	25 MB	25 MB	110 MB
Windows	MSI	30 MB	60 MB	160 MB

The following log files are available:

- installation.log — Log from ndlaunch, responsible for downloading and install all packages required for FlexNet inventory agent
- policy.log — Generated by the policy download component, mgspolicy
- schedule.log — Log from the ndschedag schedule component
- tracker.log — Generated by the inventory component, ndtrack
- uploader.log — Log from the ndupload file upload component
- usageagent.log — Generated by the mgsusageag usage tracking service.

Memory requirements

- Minimum RAM: 512 MB
- Recommended RAM: 2 GB

In general, through a cycle of inventory gathering and upload, the memory demand is in the order of 5-30 MB.

Communication protocols and ports

In the network layer, both IPv4 and IPv6 formats are supported for communications between an inventory beacon and components of the FlexNet inventory agent (specifically, `ndtrack`, `ndupload` and `ndlaunch`). (However, communications with the central application server or any component server thereof still require IPv4 across the network layer.) For more details on the architectural impacts, see Support for IPv6 Networks.

All ports used by FlexNet inventory agent are configurable to any value through preference settings, for example by including the port number in URL settings. The default values for communications supported between the FlexNet inventory agent (in the Agent third-party deployment case) and inventory beacons are:

- File upload and download using HTTP protocol: port 80
- File upload and download using HTTPS protocol: port 443
- FTP file transfers (not supported by FlexNet Beacon on inventory beacons, but may be used in custom infrastructure): port 21
- Windows direct file upload and download (UNC shares) may be used in custom infrastructure, but are not directly supported by FlexNet Beacon
- Additional ports may be required if supporting a proxy.

Supported packages to inventory

FlexNet inventory can include data from most package technologies supported by the operating systems, and some additional third-party packaging technologies:

Platform	Supported package technologies
All platforms	InstallAnywhere (IA), InstallShield Multiplatform (ISMP), BEA/Oracle Installer (BEA), Oracle Universal Installer (OUI), IBM Installation Manager (IIM)
AIX	LPP, RPM
HP-UX	Software Distributor SD-UX Package
Linux	RPM (Red Hat, CentOS, Oracle, SuSE, Fedora, etc), DPKG (Debian, Ubuntu).
macOS	Mac Application Bundle, Mac Package Bundle
Solaris	Sys V Package (pkg), IPS
Windows	MSI, Add/Remove Programs Registry Key

However, FlexNet inventory cannot collect data from some of the less common or newer operating system technologies and many third-party technologies. Some known examples include:

- All platforms — IBM InstallStream, IBM Tivoli Netcool Installer
- macOS — Mac flat package.

System load benchmarks

The following notes reflect observed behavior on sample systems using the full installed FlexNet inventory agent.

Task	Run duration (seconds)	CPU usage (seconds)	CPU usage (% of single core)	Memory usage	Network load
Inventory collection	13 to 240 s	5 to 130 s	10% to 50%	4 MB to 20 MB	10 KB to 200 KB per upload
Usage monitoring	Ongoing	Under 1 s / day	Negligible	4 MB to 8 MB	5 KB to 20 KB / day
Policy update*	15 s to 33 s	Under 1 s per policy	Negligible	3 MB to 5 MB	10 KB to 100 KB per policy update

* For each policy update, FlexNet inventory agent downloads its current policy from its preferred inventory beacon. This links to several packages:

- Failover settings (upload and download locations on all available inventory beacons, all of which are configured for anonymous authentication)
- Client configuration (the preferences for use by all installed components included in the FlexNet inventory agent)
- Schedule
- Inventory settings and extensions (`InventorySettings.xml`)
- Agent self-upgrade package.

Using each package, FlexNet inventory agent checks the last downloaded content against the version currently on the inventory beacon. When there is no change, there is no further download (so that the checking process is both rapid and lightweight for the network). Changed items are downloaded, which may increase the network load for that occasion. The largest single impact is when a new agent update package is declared: for details of the download size per platform for update packages, see the disk space listing above.

Agent Third-Party Deployment: Accounts and Privileges

When you choose to deploy the FlexNet inventory agent using third-party tools under your own management, you handle all the account security required for deployment and installation on target devices. The following comments assume that installation is complete, and address only the account requirements for ongoing operation.

The operational account requirements vary slightly across platforms.

Microsoft Windows

FlexNet inventory agent runs as the local SYSTEM account.

UNIX-like platforms

The FlexNet inventory agent can operate in either of the following two modes:

- **Default operation mode:** Runs as the `root` user and requires full `root` access.
- **Least privilege operation mode:** Runs as the `flxrasvc` standard user.

Note: *Whether the default mode or the least privilege mode is running on an agent must be configured when the agent is installed or upgraded.*

If the agent has been installed for the default operation mode, it must run as `root` for all its services on the local device. If the agent has been installed for the least privilege operation mode, `sudo` must be installed on the local device and the path to the `sudo` binary must be set in the `PATH` environment variable.

The following security settings are effective:

Note: *The `/opt/managesoft` directory is the default base installation path. Your customized installation path might be different.*

- If the agent runs in the default operation mode:
  - The `/var/opt/managesoft` directory is only accessible by `root`.
  - The `/opt/managesoft/lib` and `/opt/managesoft/libexec` folders are completely locked down to root only.
  - The `/opt/managesoft/bin` folder is open to all, to allow easy access to the path of the executables in the folder when using privilege escalation tools like `sudo`.
  - The executables in the `/opt/managesoft/bin` folder are locked down to root only.
  - The `/opt/managesoft/documentation` and `/opt/managesoft/software tag` folders are readable by all.
- If the agent runs in the least privilege operation mode:
  - The `/var/opt/managesoft` directory is readable by all.

Agent Third-Party Deployment: Implementation

Preparation and third-party deployment of the FlexNet inventory agent varies across different platforms.

For all platforms, start with Agent Third-Party Deployment: Collecting the Software.

Thereafter branch, based on the operating system running on the target device:

- For deployment to Microsoft Windows, read the subtopics under Agent Third-Party Deployment: Configuring Installation on Microsoft Windows

- For deployment to UNIX-like systems, read the subtopics under Agent Third-Party Deployment: Configuring Installations on UNIX-like Platforms.

Note: *If you wish to self manage all future upgrades of FlexNet inventory agent, please ensure automatic deployment of FlexNet inventory agent is disabled in FlexNet Manager Suite. Do this by setting the* ***Update mode*** *to* ***Do no upgrade automatically*** *(for details, see the online help, under* ***FlexNet Manager Suite Help > Discovery & Inventory > Inventory Settings****). This will prevent competing attempts to upgrade to the latest version of FlexNet inventory agent by FlexNet Manager Suite, and then to downgrade FlexNet inventory agent to the version you have chosen in your own third party deployment setup.*

Agent Third-Party Deployment: Collecting the Software

You have decided to deploy the FlexNet inventory agent with a third-party tool of your choice. Start with the appropriate version(s) of the FlexNet inventory agent.

The FlexNet inventory agent is supported on a variety of platforms (listed in Agent Third-Party Deployment: System Requirements).

Before collecting your software and arranging deployment, it is best practice to ensure that there is an inventory beacon available within each subnet where you might execute discovery rules. This allows the inventory beacon to reliably use ARP or `nbtstat` requests to determine the MAC address of a discovered device (reliability of these results is reduced across separate subnets). Where, across subnets, only an IP address can be found for a device (that is, the device data is missing both a MAC address and a device name), a record is created for the discovered device; but because IP addresses may be dynamic, this is insufficient to allow merging with more *complete* records (which also contain either or both of the MAC address and a device name).

Such complete discovery records may be created automatically when inventory is first returned from the locally-installed FlexNet inventory agent: not finding an existing, complete and matching discovered device record to link with the inventory device record, FlexNet Manager Suite automatically creates one. This means you may see multiple discovered device records with duplicate IP addresses: one record is complete (from inventory), and one or more others are missing identifying data (across subnets) as discussed. These cannot be merged automatically, and you are left with a manual task to clean up incomplete duplicate discovered device records. What's worse, if you have a rule to repeat the discovery process (for example, looking for newly-installed devices) and you still have incomplete discovery data from an inventory beacon reaching across subnet boundaries, the unmatched and incomplete record is recreated at each execution of the discovery rule.

In contrast, having a local inventory beacon in the same subnet as target devices provides both the IP address and the MAC address, which is sufficient for matching discovered device records. If you must do discovery across subnet boundaries without a local inventory beacon, ensure that there are full DNS entries visible to the inventory beacon for all devices you intend to discover. This allows the inventory beacon to report both an IP address and a name (either the device name or a fully-qualified domain name [FQDN]), which combination is again sufficient for record matching.

When your network infrastructure is in place, you can begin to deploy the FlexNet inventory agent.

To collect the FlexNet inventory agent software:

1. On the workstation where you plan to configure the package for installing FlexNet inventory agent, use a web browser to log in to the web interface for FlexNet Manager Suite.

2. Navigate to **Discovery & Inventory > Inventory Settings**.

   The **Inventory Settings** page displays.

3. Scroll down to the **Inventory agent for download** section.

4. Prepare for editing the bootstrapping file:

   - For target devices running any version of Microsoft Windows: click the hyperlink to **Download bootstrapping template file**, saving the file to a folder of your choice (such as `C:\temp`). This file must be customized for your deployment, minimally to identify the bootstrap inventory beacon with which FlexNet inventory agent will first communicate (details are forthcoming in Agent Third-Party Deployment: Edit the Configuration File for Microsoft Windows).

   - For target devices running UNIX-like systems: there is no equivalent download of a sample file, but full details for constructing your own are included in Agent Third-Party Deployment: Configuring the Bootstrap File for UNIX. No immediate action on the bootstrapping file is needed now.

5. From the **Inventory agent** drop-down list, select the (first) version of the FlexNet inventory agent you want to deploy.

   Be sure to scroll down, as the most recent releases are at the bottom of the drop-down list. You may choose whichever recent version is approved for use in your enterprise. In general, it is recommended to deploy the latest version.

   Tip: *If you are configuring a UNIX-like server for collection of Oracle inventory by the installed FlexNet inventory agent, be sure to download version 13.2.0 or later. Earlier releases of the FlexNet inventory agent may fail to collect Oracle inventory on servers where permissions prevent global access to Oracle directories or files.*

   The same software is known by different (historical) names on different platforms (a managed device is one which has the FlexNet inventory agent locally installed). In the current release, you can select from the following supported platforms:

   - FlexNet Inventory Agent, which is for Windows platforms

     Tip: *In earlier releases, the FlexNet inventory agent was known as ManageSoft for Managed Devices.*

   - ManageSoft for AIX Managed Devices
   - ManageSoft for HP-UX Managed Devices
   - ManageSoft for Linux (i386) Managed Devices
   - ManageSoft for Linux (x86_64) Managed Devices
   - ManageSoft for Mac OS X Managed Devices
   - ManageSoft for Solaris (sparc) Managed Devices
   - ManageSoft for Solaris (x86) Managed Devices.

6. Click **Download**, and save the archive to a folder of your choice.

7. If necessary, repeat the download of any additional versions that you choose to configure and deploy.

Agent Third-Party Deployment: Configuring Installation on Microsoft Windows

Broadly, there are two ways to configure the installation of the FlexNet inventory agent for Microsoft Windows:

- You can edit the Windows Installer command line to add transforms or specify that particular components should or should not be installed. For more details, see Agent Third-Party Deployment: Customizing the Windows Installer Command Line.
- You can set preferences in the bootstrap configuration file, as described in Agent Third-Party Deployment: Edit the Configuration File for Microsoft Windows.

If you are using alternative deployment methods but then wanting the managed device to fit neatly into the standard operating procedure of FlexNet Manager Suite, there is nothing more to do after those steps. In contrast, if you are customizing the installation or behavior of the FlexNet inventory agent, you could also review Agent Third-Party Deployment: Protecting Your Customizations in case you want to prevent your customizations being over-written by the system-wide standard policy. When you are satisfied with both forms of configuration/customization and your protection settings, you can use your preferred deployment tool to distribute the completed packages for installation on target devices.

Agent Third-Party Deployment: Edit the Configuration File for Microsoft Windows

Initial implementation of the FlexNet inventory agent is controlled by its bootstrapping configuration file.

In Agent Third-Party Deployment: Collecting the Software, you downloaded the template for the bootstrap configuration file (`mgssetup.ini`) for use on Microsoft Windows platforms. This file can contain many legacy settings, especially for those interested in customizing the behavior of the FlexNet inventory agent. For example, the latter part of the file allows for individual preferences to be set that will automatically be written to the registry of the managed device during the installation process (some of these are documented in the chapter on Preferences for the advanced administrator).

However, for a straightforward deployment and installation, there are only two (or perhaps three) significant specifications required:

- Identifying the inventory beacon that the FlexNet inventory agents should contact after installation, to collect their policy (which includes rules to apply, schedules, and the like) –– this much is mandatory
- Configure whether or not these managed devices should monitor application usage (by default they do not)
- If you are in an environment using IPv6 in your network layer, you may choose to prefer IPv6 formats for communications between an inventory beacon and components of the FlexNet inventory agent.

Some sections of the bootstrap file are historical, and should not be modified (as noted below).

To modify the `mgssetup.ini` configuration file:

1. Open a copy of `mgssetup.ini` in a flat text editor of your choice.

   As you may need several versions of your custom bootstrap files, it is best practice to save a backup copy of the

original, unedited file. In deployment, the file name must not be changed. It is therefore good practice to save each customized copy in its own directory. Also beware of editors which may insert special (non-ASCII) characters in the file.

Note: *Lines starting with a semi-colon character (;) are comments and have no effect. 'Uncommenting' a line means to remove the semi-colon from the start of the line.*

2. To customize the installation directory for the FlexNet inventory agent, locate the lines (around line 42) that give the default path, and modify the value (such as in this example to switch to E: drive):

```
; Set product installation directory.
TMPMAINDIR = E:\Program Files (x86)\Flexera Software\Agent
```

3. To cause the installed FlexNet inventory agent to immediately seek its policy and commence normal operation, ensure that the following line is *not* commented out:

```
INSTALLMACHINEPOLICY = 1
```

By default, when policy is downloaded and installed, a user interface may be presented. To remain in quiet mode and suppress this user interface, ensure that the following line (several lines further down) is uncommented, so that the `UILevel` variable is set to `Quiet`:

```
; Additional arguments to policy client.
POLICYARGS = -o InstallDefaultSchedule=True -o UILevel=Quiet
             -o PostponeUserInteractionLevel=Quiet -o PostponeByDefault=False
```

Tip:

- *Without the `INSTALLMACHINEPOLICY` setting (either in the `mgssetup.ini` file or added to the installation command line), a successful installation ends with the FlexNet inventory agent in place but doing nothing, awaiting delivery of a policy by some third-party means.*
- *In contrast, when this preference is correctly set as above, after installation the FlexNet inventory agent immediately attempts to contact its inventory beacon (specified in the following preference) to collect its policy; and if successful, to commence operations in line with the policy settings.*

(If your `mgssetup.ini` file still contains mention of a user policy, ensure that this preference remains commented out, as user-based policy is no longer supported.)

4. To specify the inventory beacon from which this managed device should collect its initial policy, locate the following, and uncomment and customize the second line, following the guidelines below:

```
[Custom Install Settings]
DEPLOYSERVERURL = http://beacon.server.com/ManageSoftDL
```

Tip: *The legacy naming is because an inventory beacon is derived from an earlier form of deployment server, and still deploys policy and agent updates to its managed devices.*

- The trailing `ManageSoftDL` is mandatory, and identifies the web share on the inventory beacon which must be accessible to managed devices (check permissions).

- The value must be a URL (a UNC value of the form `\\servername\ManageSoftDL$` will not work). The URL may have either of the following protocols:
  - `http://`
  - `https://`
- The *`beacon.server.com`* placeholder may be replaced with a host name, a fully-qualified domain name (FQDN), or an IP address in either IPv4 or IPv6 format. (Keep in mind that the IPv6 format is supported because it is specified at the FlexNet inventory agent end of the communication link; and that specifications made at the inventory beacon end cannot use IPv6 format because it is not supported by legacy versions of the FlexNet inventory agent that also receive content distributed from inventory beacons.)
- The URL must be no longer than 80 characters.

Important: *If this value is not set correctly, devices will not be managed and will require a software re-install.*

Remember that the `INSTALLMACHINEPOLICY` preference must be true (see previous step).

If the policy was not initially available (perhaps because of network issues, or because the inventory beacon was unavailable when the FlexNet inventory agent attempted to connect, or because policy had not yet been prepared and distributed from the central application server), FlexNet inventory agent retries policy collection at each reboot of the managed device until successful. (This behavior is different than for the FlexNet inventory agent on UNIX-like systems, where it makes a daily attempt to download policy at a random time each day.)

5. To switch the default behavior for application usage tracking by the FlexNet inventory agent from disabled to enabled, locate and uncomment the following line:

```
[Usage Agent]
USAGEAGENT_DISABLE = False
```

Background: Application usage tracking (sometimes called "application metering") by the FlexNet inventory agent is disabled by default. When enabled, it works by tracking the time during which installed files are opened and active on the targeted devices, where those installed files are known to be part of a particular installed application. This usage data is uploaded to the central application server, where the usage tracking calculations occur at each license reconciliation (whether or not usage data is available from any particular inventory source). The `USAGEAGENT_DISABLE` setting from the `mgssetup.ini` file is written to the registry on all adopted devices in *`[Registry]`*`\ManageSoft\Usage Agent\CurrentVersion\Disabled`, with the default being `True` (so that usage tracking in inventory is disabled). Independently, you can also control the same usage tracking preference through the web interface, in the target settings for any discovery and inventory rule (navigate to **Discovery & Inventory > Discovery and Inventory Rules > Targets** tab, and scroll down to the **Application usage options** section). This means that a good practice is to adopt devices with usage tracking defaulting to disabled, and then selectively turn it on for your desired targets.

6. To prefer IPv6 format for IP addresses (when the installed FlexNet inventory agent will operate in a network segment that uses this format), add the last two lines to this section for creating registry entries in `Common`:

```
;============================================================================
; Registry settings to be created under
```

```
; HKLM\Software\ManageSoft Corp\ManageSoft\Common
[Common]
;desc0 = Customization\Testing\TestValueName
;val0  = TestValueValue
; desc1 =
; val1  =
; desc2 =
; val2  =
; ... etc.
desc0 = PreferIPVersion
val0 = IPv6
```

Where a DNS returns both IPv4 and IPv6 addresses when resolving a server name, this setting causes the installed FlexNet inventory agent to give first preference to the IPv6. If the setting is not specified, the default is to prefer the IP version of the first address returned by the DNS (possibly as amended by the operating system on the local device). The setting is placed in `Common` because it is used by several components of the FlexNet inventory agent, including the tracker, the launcher, and the uploader.
For most situations, you have finished configuring this file. You may skip the advanced steps, and see the notes at the end.

7. ADVANCED USE ONLY: Optionally, insert any preferences for the operation of the FlexNet inventory agent in the later sections of the `mgssetup.ini` file that allow storing the preferences in the Windows registry.

   For example, if you wanted to modify the current default values (shown here as an example) for the fail-over algorithms that determine how the FlexNet inventory agent attempts to prioritize inventory beacons when choosing which one to access, edit the following section, changing to your own preferences:

   ```
   ; Registry settings to be created under
   ; HKLM\Software\ManageSoft Corp\ManageSoft\NetSelector
   [NetSelector]
   desc0 = SelectorAlgorithm
   val0  = MgsRandom;MgsPing;MgsADSiteMatch
   ```

   More more information about the available preferences, see Preferences.

   ***Note:** Other settings in the `mgssetup.ini` file should typically be left unchanged. They may be edited for advance, legacy, or other unusual configuration scenarios. If you need such changes, follow the guidance provided within the template file itself.*

8. Save the customized copy of `mgssetup.ini` in the same folder as the FlexNet Inventory Agent zip archive downloaded in Agent Third-Party Deployment: Collecting the Software.

   (If you are preparing distinct customizations for different parts of your computing estate, use folder naming to separate and distinguish the different copies of `mgssetup.ini`. Do not rename the `mgssetup.ini` file.)

***Tip:** For another advanced preference, see Agent Third-Party Deployment: Uninstalling from Windows Platforms.*

The following legacy sections of the configuration file should not be edited, and should be left commented out:

- `;BOOTSTRAPPEDPOLICY`

- ;WAITFORPOLICY
- ;IGNOREPOLICYFAILURE.

Agent Third-Party Deployment: Customizing the Windows Installer Command Line

You can customize the Windows Installer in the usual ways.

The two paths for configuring your installation of the FlexNet inventory agent are:

- Configuring the mgssetup.ini file (for which see Agent Third-Party Deployment: Edit the Configuration File for Microsoft Windows)
- Modifying the MSI installation, covered in this topic. This entire process is optional, for advanced use by those who have transforms planned.

You downloaded the installer archive in Agent Third-Party Deployment: Collecting the Software, saving it to a convenient directory. Return to that location now.

To customize the Windows Installer command line:

1. Unzip the downloaded archive (named like managesoft-*MM.N.B*.zip, where the placeholders represent release numbering).

   The MSI file for the FlexNet inventory agent is called FlexNet Inventory Agent.msi. You can customize the installation, for example, by installing or preventing installation of particular components, or by applying transforms. To do this, you can edit the Windows Installer command line, stored in the Setup.ini file that is in the same folder as the MSI. (Alternatively, you may prefer to run the Windows Installer from the command line.)

2. In the unzipped archive, open Setup.ini in the flat text editor of your choice.

3. Locate the following section, and amend the command line as required:

   ```
   [Startup]
   CmdLine=/l*v "%TEMP%\FlexNet Inventory Agent.msi.log"
   ```

   For example:

   - To apply a transform, append TRANSFORMS="custom.mst" to the end of the command line.
     Use semicolons (;) to separate multiple transforms. For example, to apply custom1.mst and custom2.mst for both initial installs and upgrades, the addition would look like:
     TRANSFORMS="custom1.mst;custom2.mst"

   - Add or remove any other Windows Installer command line options.
     For example, to prevent installation of the application usage agent on computers being brought under management, add REMOVE=aua to the end of the CmdLine:

     ```
     [Startup]
     CmdLine=/l*v "%TEMP%\FlexNet Inventory Agent.msi.log" REMOVE=aua
     ```

4. Save the modified `Setup.ini` file.

When you have completed modifications to the configuration file and to the command line of the installer, you can use your preferred deployment tool(s) to pack, validate, and deploy the installation package. See the following deployment notes.

Agent Third-Party Deployment: Deployment Notes for Windows Platforms

You have finished editing the bootstrap configuration file (`mgssetup.ini`) and, if necessary, customized the MSI command line stored in `Setup.ini`. These files may now be handed off, together with `FlexNet Inventory Agent.msi` and the associated `Data1.cab` file, for packaging and deployment. The following notes about the installation may be helpful to the packaging team.

1. The `mgssetup.ini` bootstrap configuration file must be placed in the same folder as the `FlexNet Inventory Agent.msi` file before executing the install command.
2. In this location, the `mgssetup.ini` file must be readable by both the SYSTEM and installing user at installation time. This means that the MSI package cannot be installed from a mapped drive (which would only be visible to the installing user), or a file share which is not visible to both user accounts.
3. As usual, the installing account needs administrator privileges to complete the installation.
4. If you are using Microsoft Installer and providing a command line, you may conveniently modify the installation directory like this (all in one line):

   ```
   msiexec  /i “FlexNet Inventory Agent.msi”
        INSTALLDIR=”E:\Program Files (x86)\Flexera Software\Agent” /qb
   ```

   If you are not preparing your own command line, see Agent Third-Party Deployment: Edit the Configuration File for Microsoft Windows for another method of changing the installation directory.
5. For operation, the FlexNet inventory agent services must be configured to run as the Local System account.
6. Once the FlexNet inventory agent has been successfully deployed and is reporting inventory, the device is visible in the **Discovery & Inventory > All Discovered Devices** page. With Agent third-party deployment, devices may not appear in this list until you run a license reconciliation process, as previously undiscovered devices are automatically added here when their reported inventory is processed.

Tip: *The* ***Agent installed*** *column on that page is set to* `Yes` *only when:*

- *The discovered device was "adopted" (through the automated processes where an inventory beacon installed the standard FlexNet deployment package) within the last 30 days, creating a grace period within which inventory uploads are expected*
- *The installed FlexNet inventory agent uploads inventory, within which the agent itself can be recognized, in cycles of 30 days or less.*

Agent Third-Party Deployment: Uninstalling from Windows Platforms

If you decide to remove the FlexNet inventory agent from a system running Microsoft Windows, you have two basic approaches:

- Use the deployment tool with which you completed the original installation to now remove it.
- Manually navigate to Add/Remove Programs (or equivalent) in Microsoft Windows and uninstall FlexNet inventory agent.

Tip: *This standard Windows facility means that, depending on privileges in your environment, users of managed devices can remove FlexNet inventory agent. If you wish to prevent users on these target devices from using Windows Add/Remove Programs to uninstall the FlexNet inventory agent, uncomment this line in* `mgssetup.ini` *before deployment:*

```
   [Custom Install Settings]
ALLOWUNINSTALL = 0
```

Of course, this reduces your options for uninstallation, and you must then use your deployment tool to do any planned removal.

Uninstalling the agent by means of manual uninstall, third party deployment uninstall, or auto-upgrade, will remove the executable files but leave the following behind:

- Registry data added by policy or the agent itself (`HKLM\Software\ManageSoft Corp, HKCU\Software\ManageSoft Corp`)
- Agent output files (`C:\ProgramData\ManageSoft Corp`)
- Agent logs (`C:\Windows\temp\ManageSoft, %USERPROFILE%\AppData\Local\Temp\ManageSoft`)
- Policy data (`C:\Program Files (x86)\ManageSoft\Launcher`)
- Folders `%ProgramFiles%\ManageSoft\Launcher\Cache`, and its peer `PkgCache`, since these may contain content that a future re-install of the FlexNet inventory agent can use. (On 64-bit platforms, substitute `%ProgramFiles(x86)%.`)

Note: *Anything created by the agent at runtime will typically not be removed when uninstalling the agent because the data is unknown to Windows Installer.*

Agent Third-Party Deployment: Configuring Installations on UNIX-like Platforms

This section includes information on preparing and deploying your bootstrap configuration file, and installing the FlexNet inventory agent on UNIX-like platforms.

You can choose to install the FlexNet inventory agent for either the default operation mode or the least privilege operation mode. For more details, see Agent Third-Party Deployment: Configuring the Operation Mode on UNIX.

Agent Third-Party Deployment: Configuring the Bootstrap File for UNIX

The initial configuration of the FlexNet inventory agent can be set for UNIX-like platforms, even though no template file is provided.

For UNIX and OS X, there is no sample bootstrap configuration file available through the central application server. Instead, you can prepare your customized bootstrap configuration file as follows:

To prepare a `mgsft_rollout_response` file:

1. Copy the sample text from Agent Third-Party Deployment: Sample UNIX Bootstrap Configuration File into your preferred flat-text editor on a UNIX-like platform.

   Tip: *Do not edit the file on a Windows device, as this introduces line-ending character pairs that are invalid for UNIX, and is also likely to add an inappropriate file type.*

2. Locate and edit the following line to identify the inventory beacon from which the new managed device should download its initial policy:

   ```
   MGSFT_BOOTSTRAP_DOWNLOAD=http://beacon.mydomain.com:8080/ManageSoftDL/
   ```

   - For comparison, in the automated adoption process (the Adopted case, where the inventory beacon installs FlexNet inventory agent by remote execution), it is *mandatory* to use the HTTP protocol. Because you are independently managing your own deployment, it's also normal to use the HTTP protocol for bootstrapping, because it is simpler to set up and get operational. However, if you require the HTTPS protocol for your own deployment, insert it in this value.
   - Replace the placeholder *`beacon.mydomain.com`* with the fully qualified domain name of the inventory beacon. If required, and provided that you are using the HTTP protocol, you may instead use the server's IP address. (There are widely publicized issues around using an IP address with the HTTPS protocol.) Because you are specifying this address at the FlexNet inventory agent end of the communication link, this may use either the IPv4 or IPv6 address families. Keep in mind that, because the fail-over list of inventory beacons delivered through policy must use names (host or FQDN) to support legacy versions of FlexNet inventory agent, names are used rather than IP addresses as soon as policy is delivered.
   - If you are using the default port (80 for HTTP, and 443 for HTTPS), you can omit the port number. For any custom port numbers, include the port number in the URL as shown (`:8080`).
   - The string literal `ManageSoftDL` is the name of the web service that handles downloads to managed devices. This value is mandatory.

3. Following those same guidelines, edit the following value for the upload location on the same inventory beacon.

   ```
   MGSFT_BOOTSTRAP_UPLOAD=http://beacon.mydomain.com:8080/ManageSoftRL/
   ```

   To bootstrap the UNIX agents, both the download and upload locations must be specified. (This is not the case for the agents on Windows, where only the download location is required.) Notice that `ManageSoftRL` is the name of a web service on the inventory beacon that receives the uploaded inventory and saves it by default to

%CommonAppData%\Flexera Software\Incoming\Inventories.

Tip: *The section about proxies in the bootstrap file is only required in the unusual circumstances that you have a proxy between the managed device(s) and the inventory beacon(s) (in which case follow the guidance in the template). When this is not the case, leave these settings commented out.*

4. Optionally, configure the local web server on inventory beacons to use HTTPS protocol.

   The web server on the inventory beacon defaults to using the HTTP protocol for simplicity of communications between managed devices and the inventory beacon. However, if you need to use the HTTPS protocol over this leg of the upload/download chain, you may also need to configure how the managed devices should check the security certificates originating from the inventory beacon server. The choice of protocol, along with the configuration for certificate checking if HTTPS is used, are downloaded to managed devices as part of their policy (policy is generated automatically by the inventory beacons). From large to small granularity, the available certificate controls that can be configured in the `mgsft_rollout_response` file include:

   - Whether to check the security certificates at all.
   - If checking the supplied certificate, whether to check that the certificate is still current (that is, checking that the certificate has not been revoked by a certificate authority). The default is to validate that the certificate has not been revoked and is still current. This is particularly important when using certificates from public certificate authorities on the Internet. Perhaps if you are providing your own internal certificate authority and long-term certificates, you may turn off the check for revocation of certificates.
   - Choosing between, and prioritizing, the two methods for checking certificate revocation.
   - Creating caches where downloaded revocation responses can be saved for a limited time.
   - Setting cache time-out values for each method used.

   Tip: *If you are checking server certificates, you must deploy a copy of the appropriate certificate to each managed device. This allows the managed device to check the supplied certificate that covers each download from the inventory beacon server. This is described in Agent Third-Party Deployment: Installing FlexNet inventory agent on UNIX, and there is more information in Agent Third-Party Deployment: Enabling the HTTPS Protocol on UNIX Agents and Agent Third-Party Deployment: HTTPS CA Certificate File Format (UNIX).*

   Settings declared in the `mgsft_rollout_response` affect all components of the FlexNet inventory agent equally. It is also possible to override behaviors for individual components. For details see the preference topics included in the following list. To modify the defaults for certificate checking, use the following settings (in the order corresponding to the above descriptions):

   a. Server certificates are checked by default. Uncomment and edit the following line to prevent any certificate checking:

      ```
      MGSFT_HTTPS_CHECKSERVERCERTIFICATE=false
      ```

      With this setting false, you get the standard encryption of network traffic between managed device and inventory beacon, but no further security. (After installation of the FlexNet inventory agent, this setting appears as `CheckServerCertificate` in the `/var/opt/managesoft/etc/config.ini` file, in the `[ManageSoft\Common]` section. See CheckServerCertificate for more.)

b. The client certificate for mutual TLS is not used by default. To support mutual TLS and allow the client (inventory device) to present the server (inventory beacon) with a certificate, uncomment and edit the following line:

```
MGSFT_HTTPS_ADDCLIENTCERTIFICATEANDKEY=true
```

With this setting true, mutual TLS authentication is possible, and a client-side certificate and key are required. (After installation of the FlexNet inventory agent, this setting appears as `AddClientCertificateAndKey` in the `[ManageSoft\Common]` section of the `/var/opt/managesoft/etc/config.ini` file. For more information, see AddClientCertificateAndKey.)

c. Optionally when you are using internal certificate authorities, you may uncomment and edit the following line to prevent a check for revocation of certificates:

```
MGSFT_HTTPS_CHECKCERTIFICATEREVOCATION=false
```

With this setting false, you get a check that the download is coming from the genuine inventory beacon; but there is no check whether the inventory beacon may have been compromised and its certificate subsequently revoked. (After installation of the FlexNet inventory agent, this setting appears as `CheckCertificateRevocation` in the `/var/opt/managesoft/etc/config.ini` file, in the `[ManageSoft\Common]` section. See CheckCertificateRevocation for more.)

d. Optionally, modify the method(s) that the FlexNet inventory agent uses to check whether a downloaded server certificate has been revoked by a certificate authority. Uncomment and edit this line:

```
MGSFT_HTTPS_PRIORITIZEREVOCATIONCHECKS=OCSP,CRL
```

With this default setting, the FlexNet inventory agent first tries for an efficient OCSP response about the single certificate. If this fails, it next tries to download a Certificate Revocation List (CRL) from the certificate authority; but as this file lists every revoked certificate, can be a large file that is time-consuming to fetch. Reverse the order (`CRL,OCSP`) to change the priorities around; or omit one or the other (and the comma) to turn off that kind of revocation checking. (After installation of the FlexNet inventory agent, this setting appears as `PrioritizeRevocationChecks` in the `/var/opt/managesoft/etc/config.ini` file, in the `[ManageSoft\Common]` section. See PrioritizeRevocationChecks for more.)

e. Optionally, change the settings for each cache you may use by uncommenting and editing the appropriate lines from the following pair:

```
MGSFT_HTTPS_SSLCRLCACHELIFETIME=64800
MGSFT_HTTPS_SSLOCSPCACHELIFETIME=64800
```

After installation of the FlexNet inventory agent, these settings also appears in the `/var/opt/managesoft/etc/config.ini` file, in the `[ManageSoft\Common]` section. For more information about these settings, see:

- SSLCRLCacheLifetime
- SSLOCSPCacheLifetime.

5. When deploying the FlexNet inventory agent into a subnet that uses IPv6 addresses in the network layer, uncomment the following line to cause these to be used in preference to any IPv4 addresses that may also be

returned from a DNS:

```
PREFERIPVERSION=ipv6
```

This setting is used in common by multiple components of the FlexNet inventory agent (including the tracker, the launcher, and the upload component). Where this is specified but IPv6 addresses are not provided, operations fail over to the use of IPv4 addresses. Where the preference is not specified (or is specified with an unrecognized value), the default behavior is to use the IP version of the first address in the list returned from the Dynamic Name Server (DNS) through the operating system (which, depending on local settings, may also affect the order of the list).

6. If you are planning to deploy the FlexNet inventory agent to a custom location on the AIX operating system, and you want to use a custom folder for data exchange by the various components, append the following line to your file:

```
COMMONAPPDATAFOLDER=/absolute/path/and/folder
```

The path should *not* contain white space characters. Use an absolute path in its simplest canonical form, without relative path elements. For example, to use the folder `/var/lib/flexera` as the data directory accessed by all components of the FlexNet inventory agent, include this line in your `mgsft_rollout_response` file:

```
COMMONAPPDATAFOLDER=/var/lib/flexera
```

Unlike the installation path, the data path is created by the installer if it does not already exist. If you omit this option from the `mgsft_rollout_response` file for a new installation, the default (`/var/opt/managesoft`) is used for the data folder. This setting is required only on the AIX platform, and only when you require a custom data folder. The setting is ignored for all other platforms.

7. If you prefer that UNIX-like devices report themselves as present in a Windows domain (which may help resolve inventories from multiple sources, as well as providing consistent data presentation in the web interface of FlexNet Manager Suite), you can set the domain name by adding lines like the following to your file:

```
# Dummy domain name for reporting by UNIX-like devices
MGSFT_DOMAIN_NAME=mydomain.com
```

Replace the *mydomain.com* placeholder with the domain name to use for reporting. (After deployment, this value is stored in the `ComputerDomain` preference, saved for UNIX-like devices in the `/var/opt/managesoft/etc/config.ini` file. For details, see ComputerDomain.)

8. Configure whether the agent will be installed for the default operation mode or the least privilege operation mode.

   - **Default operation mode:** The installed agent will operate as the `root` user and requires full `root` access. This is the default setting. To enable this mode, make sure the line about `FLEXERA_LEAST_PRIVILEGE_AGENT` is either deleted or commented out.

   - **Least privilege operation mode:** The agent will operate as the `flxrasvc` standard user. If you install the agent for this mode, make sure that `sudo` is installed on each local device and the path to the `sudo` binary is set in the `PATH` environment variable *before or at the same time* the agent is installed. To enable this mode, add the following line:

```
FLEXERA_LEAST_PRIVILEGE_AGENT=1
```

Important: *After an agent is installed for the least privilege operation mode, that agent cannot be changed back to the full privileged default operation mode. You will have to uninstall the agent and re-install the agent for the default operation mode.*

9. Save the file as `mgsft_rollout_response`.

Tip: *Leave `MGSFT_RUNPOLICY=1` unchanged, so that downloaded policy is applied after installation. For as long as policy is not available for any reason, on UNIX and OS X the agents run a daily check for policy at a random time between 8am and 11pm (local time on the managed device) until policy is successfully downloaded. (This catch-up behavior is different than the Windows agents, which rely on a machine reboot to check again for missing policy.) Once policy (with schedule) is initially downloaded, it is updated daily on the downloaded schedule, refreshing client settings, inventory-gathering schedule, and the like.*

10. Configure your preferred deployment technology to install a copy of this file as `/var/tmp/mgsft_rollout_response` on the target device(s).

    The path and file name are mandatory. This file must be present before FlexNet inventory agent is installed. Post installation scripts in the installation package for FlexNet inventory agent use properties from this file to create the initial configuration.

Tip: *In preparing the Windows bootstrap file (`mgssetup.ini`), you could turn application usage tracking on for the managed devices using the bootstrap file. This is not possible in the bootstrap file for UNIX-like systems. To turn on usage tracking, the simplest path is to set usage tracking as part of defining targets (in the web interface of FlexNet Manager Suite), so that managed devices receive this setting as part of their downloaded policy. Manually editing `config.ini` for UNIX-like platforms is also possible (see Agent Third-Party Deployment: Updating config.ini on a UNIX Device), but this approach is not as easy to scale.*

Agent Third-Party Deployment: Sample UNIX Bootstrap Configuration File

```
# The initial download location(s) for the installation.
# For example, http://myhost.mydomain.com/ManageSoftDL/
# Refer to the documentation for further details.
MGSFT_BOOTSTRAP_DOWNLOAD=http://beacon.mydomain.com:8080/ManageSoftDL/

# The initial reporting location(s) for the installation.
# For example, http://myhost.mydomain.com/ManageSoftRL/
# Refer to the documentation for further details.
MGSFT_BOOTSTRAP_UPLOAD=http://beacon.mydomain.com:8080/ManageSoftRL/

# For subnets using IPv6, uncomment to cause the inventory agent
# to prefer IPv6 addresses when both formats are returned.
# Fails over to IPv4 addresses when IPv6 is not available.
# The default behavior when this setting is not specified
# uses the IP version of the first address returned by the DNS and OS.
# PREFERIPVERSION=ipv6
```

```
       # The initial proxy configuration.  Uncomment these to enable proxy
configuration.
       # Note that setting values of NONE disables this feature.
       # MGSFT_HTTP_PROXY=http://webproxy.local:3128
       # MGSFT_HTTPS_PROXY=https://webproxy.local:3129
       # MGSFT_NO_PROXY=internal1.local,internal2.local

       # Check the HTTPS server certificate's existence, name, validity period,
       # and issuance by a trusted certificate authority (CA).  This is enabled
       # by default and can be disabled with false.
       # MGSFT_HTTPS_CHECKSERVERCERTIFICATE=true

       # Provide the client side certificate and private key for mutual TLS
       # authentication support.
       # This is disabled by default and can be enabled with true.
       # MGSFT_HTTPS_ADDCLIENTCERTIFICATEANDKEY=false

       # Check that the HTTPS server certificate has not been revoked. This is
       # enabled by default and can be disabled with false.
       # MGSFT_HTTPS_CHECKCERTIFICATEREVOCATION=true

       # Prioritize the method of checking for revocation of the HTTPS server
       # certificate. You can reverse the values to swap the default order.
       # MGSFT_HTTPS_PRIORITIZEREVOCATIONCHECKS=OCSP,CRL

       # These settings control the caching of HTTPS server certificate checking.
       # Default values are shown (these take effect when no settings are
specified).
       # Lifetime is in seconds. There are parallel settings for using CRL or OCSP
       # checking. See documentation for more information.
       # MGSFT_HTTPS_SSLCRLCACHELIFETIME=64800
       # MGSFT_HTTPS_SSLOCSPCACHELIFETIME=64800

       # The run policy flag determines if policy will run after installation.
       #    "1" or "Yes" will run policy after install
       #    "0" or "No" will not run policy
       MGSFT_RUNPOLICY=1

       # Configure the agent to run as least privileged, default is to install full
privilege.
       # Enabling this configuration requires that the local sudoers be configured
to allow the agent
       # service account (flxrasvc) be allowed to launch specific tools as root to
operate
       # correctly.
       # FLEXERA_LEAST_PRIVILEGE_AGENT=1
```

Agent Third-Party Deployment: Installing FlexNet inventory agent on UNIX

You downloaded the installers of interest in Agent Third-Party Deployment: Collecting the Software. You have also customized your bootstrap configuration file (or potentially multiple variants for different platforms and contexts) in Agent Third-Party Deployment: Configuring the Bootstrap File for UNIX.

The default installation path for the complete FlexNet inventory agent on UNIX-like platforms is:

```
/opt/managesoft/bin
```

Log files for the complete FlexNet inventory agent default to:

```
var/opt/managesoft/log
```

Note: *For some UNIX-like platforms, you can customize the command line to install the FlexNet inventory agent in the folder of your choice. Because FlexNet Manager Suite uses the native installation technology on each platform, this introduces some diversity into the installation process, as described below. As well, notice the following points:*

1. *If you previously had the FlexNet inventory agent installed in the default location, and now wish to switch to a custom installation path, first uninstall the old agent.*
2. *With a customized installation path, support for in-place self-upgrade of FlexNet inventory agent requires:*
   - *The installed, running version of FlexNet inventory agent must be 13.2.0 or later, with a later version specified as the intended upgrade*
   - *The installation is on either AIX or Linux.*

   For other platforms, or when an earlier version is operational, using third-party deployment to a custom location means that you are also committing to update the FlexNet inventory agent in that location using your chosen third-party technology.
3. *If you previously turned on self-upgrade of the FlexNet inventory agent (and are now switching to a custom installation path with third-party deployment), you should typically navigate to* ***Discovery & Inventory > Settings****. In the* ***Inventory agent for automatic deployment*** *section, consider selecting* `Do not upgrade automatically` *and publishing this setting for platforms where you are implementing the custom installation path. While it is possible to use a third-party tool for deployment, and then allow in-place self-updates, this requires very careful management so that, for example, you do not allow the third-party tool to continue maintaining one installed version at the same time as the self-update policy requires a different version. A difference in policy can mean that one tool upgrades and the other tool downgrades again (or attempts to). This tussle for control can result in damaged installations and a non-functional FlexNet inventory agent.*

Currently, a custom installation folder is supported for the following platforms:

- *AIX version 5.3 or later, where the custom path is called a User Specified Installation Location (USIL)*
- *Linux x86 (RPM)*
- *Linux x86_64 (RPM)*
- *Solaris SPARC (currently does not support in-place self-upgrade)*
- *Solaris x86 (currently does not support in-place self-upgrade).*

To deploy FlexNet inventory agent to UNIX-like platforms:

1. Configure your deployment/installation tool to deliver the bootstrap configuration file to `/var/tmp/mgsft_rollout_response`.

   This file must be in place on the device before you run the installer for FlexNet inventory agent.

   Tip: *If you are installing to a custom location (USIL) on the AIX operating system, remember to include the COMMONAPPDATAFOLDER option (for details, see Agent Third-Party Deployment: Configuring the Bootstrap File for UNIX).*

2. If you have configured the bootstrap configuration file to install the FlexNet inventory agent into the **least privilege operation mode**, configure `sudo` on the target Unix system. For details, see Agent Third-Party Deployment: Configuring the Operation Mode on UNIX.

3. If the target computer device is to use the HTTPS protocol to communicate with an inventory beacon, and you require certificate checking to validate that the device is talking to the correct inventory beacon (for details, see Agent Third-Party Deployment: Enabling the HTTPS Protocol on UNIX Agents):

   a. Prepare a summary HTTPS CA certificate for the target device(s) (see notes in Agent Third-Party Deployment: HTTPS CA Certificate File Format (UNIX))

   b. Configure your deployment/installation tool to deliver the certificate file as `/var/tmp/mgsft_rollout_cert` on the target device.

      To complete certificate set-up during installation, this file must be in place on the device *before* you run the installer for FlexNet inventory agent. During installation, the `/var/tmp/mgsft_rollout_cert` file is copied to `/var/opt/managesoft/etc/ssl/cert.pem`.

      Tip: *If you do not complete this as part of the deployment and installation process, after installation you can simply copy the completed certificate to `/var/opt/managesoft/etc/ssl/cert.pem` on a device where FlexNet inventory agent is locally installed.*

   c. If you wish to use *mutual* TLS to authenticate communication between the client (inventory device) and server (inventory beacon), also configure your deployment/installation tool to deliver the *client* certificate file as `/var/tmp/mgsft_rollout_client_cert` on the target device.

      To complete certificate set-up during installation, this file must be in place on the device *before* you run the installer for FlexNet inventory agent. During installation, the `/var/tmp/mgsft_rollout_client_cert` file is copied to `/var/opt/managesoft/etc/ssl/client/client_cert.pem`.

      Tip: *If you do not complete this as part of the deployment and installation process, after installation you can simply copy the completed certificate to `/var/opt/managesoft/etc/ssl/client/client_cert.pem` on a device where FlexNet inventory agent is locally installed.*

   d. Similarly, if you are using mutual TLS, configure your deployment/installation tool to deliver the client private key file as `/var/tmp/mgsft_rollout_client_private_key` on the target device.

      To complete certificate set-up during installation, this file must be in place on the device *before* you run the installer for FlexNet inventory agent. During installation, the `/var/tmp/`

`mgsft_rollout_client_private_key` file is copied to `/var/opt/managesoft/etc/ssl/client/private/client_key.pem`.

Tip: *If you do not complete this as part of the deployment and installation process, after installation you can simply copy the certificate private key file to `/var/opt/managesoft/etc/ssl/client/private/client_key.pem` on a device where FlexNet inventory agent is locally installed.*

4. For Solaris platforms where you want a silent installation without user interaction, prepare a flat text `admin` file with the following content to include in your deployment package:

```
mail=
instance=overwrite
partial=nocheck
runlevel=nocheck
idepend=nocheck
rdepend=nocheck
space=quit
setuid=nocheck
conflict=nocheck
action=nocheck
basedir=default
```

Tip: *If you are planning a custom installation folder for your Solaris implementation, do not alter the value for `basedir` in this file. Keep the line `basedir=default` unchanged, and see the next step.*

5. If you intend to install FlexNet inventory agent in a custom location on a Solaris platform, prepare a separate package response file (such as `/var/tmp/flexera_package_response`) with the following content (noting that the variable names are case sensitive and *must* be supplied in all upper case, as shown):

```
INSTALLDIR=/install/Path
COMMONAPPDATAFOLDER=/data/Path
```

Important: *These paths must not contain white space characters. These will cause the installation to fail. Use absolute paths in their simplest canonical form, without relative path elements.*

Tip: *You may customize either or both of these paths. Omit either one for which you want to use the default path (`/opt/managesoft` and `/var/opt/managesoft` respectively).*

This file is to be included in your deployment package, and must be in place in your chosen path before the installation command line is issued.

6. Prepare the installation tool command line appropriately for each target platform.

   There are different command lines for different UNIX platforms. The following examples all assume that you execute the command line in the directory containing the installation package. For HP-UX alone, it is *mandatory* to include a fully-qualified path to the package. On other platforms, if the installation package is not in the current directory, add a value for *`[PACKAGE PATH]`* in the respective command.

   Installations on UNIX-like platforms require `root` privileges. Configure your deployment and installation

technology to execute the following commands on the appropriate platforms (substituting *[VERSION]* and *[PACKAGE PATH]* with appropriate values):

Platform	Installation command line

AIX	For the default installation path:
	`installp -aYX -d managesoft.[VERSION].bff managesoft.rte`
	 Tip: *You may use a custom data folder in conjunction with the default installation path. For more information, see Agent Third-Party Deployment: Configuring the Bootstrap File for UNIX.*
	For a custom installation path (on a single line):
	`/usr/sbin/installp -R /new/install/path -aYX -d managesoft.version.number.bff managesoft.rte`
	where
	• */new/install/path* (the User Specified Installation Location) is the base for your installation path (the path is automatically extended with `/opt/managesoft` appended to the base you provide).
	 Important: *The base path must exist on the target system before issuing the installation command line.*
	• *version.number* is the multi-part version numbering for the current installation package.
	The `-R` option creates and maintains a User Specified Installation Location (USIL) with its own package database. This has the following consequences:
	• You must use the same `-R` option for all management of the package, including upgrade of the agent with `installp`, removal of the agent with `installp`, listing the installed agent with `lslpp`, and verification of the package with `lppchk`.
	• If you are upgrading a previous installation in the default (or any other) path, and now wish to deploy to a new custom folder, first uninstall the previous installation of the FlexNet inventory agent.
	As an example command line, to install version 13.2.0 of the FlexNet inventory agent into `/u/software/opt/managesoft`, use:
	`/usr/sbin/installp -R /u/software -aYX -d managesoft.13.2.0.0.bff managesoft.rte`
	 Tip: *For future upgrades, be sure to supply the same custom installation path or USIL (with the same `-R` option) using your third-party deployment tool. You can always check your custom installation path values as follows:*

Platform	Installation command line
	`cat /etc/managesoft.ini` *As well as its installation path, the upgraded FlexNet inventory agent requires a data storage folder. The `COMMONAPPDATAFOLDER` value used for the most recent custom installation is also saved in the same `.ini` file. For a future upgrade using the same locations for executables and data, if the `mgsft_rollout_response` file is missing, the installer can use the data folder value from `/etc/managesoft.ini`. (Therefore at upgrade time, ensure that either a `mgsft_rollout_response` file or `/etc/managesoft.ini` is available, to prevent the upgraded configuration being lost.)*
Debian/Ubuntu Linux x86	`dpkg --install managesoft_[VERSION]_i386.deb`
Debian/Ubuntu Linux x86_64	`dpkg --install managesoft_[VERSION]_amd64.deb`
HP-UX	`swinstall -v -x mount_all_filesystems=false -x allow_downdate=true` `-s [PACKAGE PATH]/managesoft-[VERSION].depot managesoft`

Platform	Installation command line

Linux x86 (RPM)	For the default installation path:
	`rpm --upgrade --oldpackage --verbose managesoft-[VERSION]-1.i386.rpm`
	For a custom installation path (on a single line):
	`rpm --upgrade --oldpackage --verbose --relocate /opt/managesoft=/new/install/path --relocate /var/opt/managesoft=/new/data/path managesoft-[VERSION]-1.i386.rpm`
	Tip: *The default values to be replaced are case sensitive and must be supplied exactly as shown. The new paths must not contain any white space characters. Use absolute paths in their simplest canonical form, without relative path elements.*
	If you omit either `--relocate` option, the corresponding default value is used for the installation.
	Tip: *For future upgrades, be sure to supply the same custom installation path using your third-party deployment tool. You can always check your custom installation path values as follows:*
	`cat /etc/managesoft.ini`

Platform	Installation command line

Linux x86_64 (RPM)	For the default installation path:
	`rpm --upgrade --oldpackage --verbose managesoft-[VERSION]-1.x86_64.rpm`
	For a custom installation path (on a single line):
	`rpm --upgrade --oldpackage --verbose --relocate /opt/managesoft=/new/install/path --relocate /var/opt/managesoft=/new/data/path managesoft-[VERSION]-1.x86_64.rpm`
	Tip: *The default values to be replaced are case sensitive and must be supplied exactly as shown. The new paths must not contain any white space characters. Use absolute paths in their simplest canonical form, without relative path elements.*
	If you omit either `--relocate` option, the corresponding default value is used for the installation.
	Tip: *For upgrades, be sure to supply the same custom installation path using your third-party deployment tool. You can always check your custom installation path values as follows:*
	`cat /etc/managesoft.ini`

Mac OS X	For the default installation path (while running as root): `installer -verbose -pkg managesoft-[VERSION].pkg -target /` ***Note:*** *If you are running anti-virus software, ensure that FlexNet Manager Suite is white-listed in your anti-virus settings so that the installation is not blocked.*

Platform	Installation command line

Solaris SPARC	For the default installation path:
	`pkgadd -n -a admin -r /dev/null -d managesoft-[VERSION].sparc.pkg ManageSoft`
	Tip: *If you saved your `admin` file under a different name, modify the command line appropriately.*
	For a custom installation path:
	`pkgadd -n -a adminFile -r responseFile -d packageFile ManageSoft`
	For example (all on one line):
	`/usr/sbin/pkgadd -n -a admin -r /var/tmp/flexera_package_response -d managesoft-12.1.0.x86.pkg ManageSoft`
	Tip: *For upgrades, be sure to supply the same custom installation path using your third-party deployment tool. You can always check your custom installation path values as follows:*
	`cat /etc/managesoft.ini`

Solaris x86	For the default installation path: `pkgadd -n -a admin -r /dev/null -d managesoft-[VERSION].x86.pkg ManageSoft` For a custom installation path, use the command line shown above for Solaris SPARC.

7. You can now hand off these materials (the configured installer, the bootstrap configuration file, the optional HTTPS CA certificate, and for Solaris the `admin` file and optional package response file) to the people responsible for packaging and deployment to the target devices.

Tip: *If you are manually completing a single installation on an UNIX-like device, remember to have the bootstrap configuration file and the HTTPS CA certificate in place first, and then run the installer with your prepared command line.*

After installation (assuming the standard setting of `MGSFT_RUNPOLICY=1`), FlexNet inventory agent attempts to connect with the inventory beacon. You may need to protect your customization, for which see the notes in Agent Third-Party Deployment: Protecting Your Customizations.

Agent Third-Party Deployment: Configuring the Operation Mode on UNIX

Before the installation, you can configure to install the FlexNet inventory agent into either of the following two operation modes on the target UNIX system:

- **Default operation mode:** The installed agent will run as the `root` user and requires full `root` access.
- **Least privilege operation mode:** The installed agent will run as the `flxrasvc`standard user.

Important: *You can upgrade an existing agent that has been installed for the default operation mode to the least privilege operation mode. However, if an agent has been installed for the least privilege operation mode, you cannot change it to the full privileged default operation mode. To change an agent from the least privilege operation mode to the default operation mode, you must uninstall and re-install the agent.*

Important: *If an agent has been installed for the least privilege operation mode, it is not able to perform self-upgrade to new versions.*

Configuring for the default operation mode

By default, an agent will be installed or upgraded for the default operation mode. However, if you are upgrading an existing agent from the least privilege operation mode to the default operation mode, make sure that your bootstrap configuration file is correctly configured for the default operation mode. For details, see Agent Third-Party Deployment: Configuring the Bootstrap File for UNIX.

Configuring for the least privilege operation mode

To install the agent for the least privilege operation mode on UNIX, you need to complete the following tasks before the installation:

1. Configure the bootstrap file for the least privilege operation mode by following the instructions in Agent Third-Party Deployment: Configuring the Bootstrap File for UNIX.
2. Make sure `sudo` is installed on the target UNIX system and the path to the `sudo` binary is set in the PATH environment.
3. Configure `sudo` on the target UNIX system to grant privilege to the required tools according to the following table. For a sample `sudoers` file to configure `sudo`, see Agent Third-Party Deployment: Sample Sudoers File.

   Important:

   - *Always use `visudo` to edit and verify a `sudoers` file, and ensure the changes do not corrupt the `sudoers` file which can cause `sudo` to be inoperable.*
   - *Some systems (notably Red Hat) have a sudoers entry `Defaults requiretty` that prevents running `sudo` when no `tty` is present. The agent requests `sudo` to run without any prompts, but this entry still causes `sudo` to fail without a `tty`, which will result in the agent being unable to launch necessary utilities through `sudo` even though the sudoers file has granted them the access. To avoid this issue, the `Defaults requiretty` entry must be removed from the sudoers file. For more information, see Bug 1020147 on Red Hat Bugzilla.*

   The following table provides information about which tools require `sudo` privilege on which platforms for an

agent running in the least privilege operation mode.

Tool/Path	Tool function	Platform requiring sudo privilege	Consequence of missing sudo privilege
Non-Flexera tools			
`/sbin/ifconfig` or `/usr/sbin/ifconfig`	Obtain network interface data	Solaris	Network interface data missing
`/usr/sbin/dmidecode`	Obtain hardware serial number	Linux x86 Solaris x86	Serial number missing
`/usr/sbin/ioscan`	Obtain local disk drive data	HP-UX	Disk drive data missing
`db2licm` ***Note:*** *The path to `db2licm` depends on where DB2 is installed. That path needs to be added to sudoers.*	Launched to obtain inventory for IBM DB2	All supported Unix platforms	IBM DB2 inventory missing or incomplete
Flexera agent tools			
Note: *Paths to Flexera agent tools are dependent on where the agent is installed. The default base installation path is `/opt/managesoft/`. Your custom installation path might be different if your platform supports custom installation path. For a list of platforms that support custom installation path, see Agent Third-Party Deployment: Installing FlexNet inventory agent on UNIX.*			
`/opt/managesoft/libexec/flxecmc`	Obtain hardware properties used to generate and validate the local agent ID	Linux ***Note:*** *Sudo privilege for flxecmc should not be revoked once it has been provided on any given Linux machine. Doing so will cause the agent ID to not be reported even if the machine currently has a valid ID.*	Agent ID not generated or an existing ID not validated

Tool/Path	Tool function	Platform requiring sudo privilege	Consequence of missing sudo privilege
/opt/managesoft/libexec/flxfsscan	Run file system scan and provide ability to read file data and metadata	All supported Unix platforms	Data missing on inventory reliant on file scan (Oracle, Oracle FMW, Java, Jboss, supported installation evidence types)
/opt/managesoft/libexec/flxoracleinv	Provide impersonation for running Oracle database queries scripts used in obtaining Oracle inventory	All supported Unix platforms	Oracle database inventory missing
/opt/managesoft/libexec/flxping	Implement custom ping support for MgsPing net selector algorithm	All supported Unix platforms	MgsPing net selector algorithm failing to rank multiple beacon connections
/opt/managesoft/libexec/flxps	Obtain currently running process data	All supported Unix platforms	Data missing on inventory reliant on running process data (usage, Oracle, Oracle FMW, Java, and IBM MQ)
/opt/managesoft/libexec/flxsysinfo	Obtain local disk drive data	Linux HP-UX	Disk drive data missing

For more details about the least privilege operation mode, such as what happens at installation, what processes are launched and how to run agent component directly after the installation, see Agent Third-Party Deployment: Least Privilege Operation Mode.

Agent Third-Party Deployment: Least Privilege Operation Mode

This topic provides information about how the least privilege operation mode works differently from the default operation mode. For instructions about how to configure the agent installation for either the least privilege operation mode or the default operation mode, see Agent Third-Party Deployment: Configuring the Operation

Mode on UNIX.

In this topic, you can find the following information:

- What happens at installation
- The Flexera agent service account
- Additional processes launched by inventory collection
- How to run agent components directly

What happens at installation

When the agent is configured at installation for the least privileged operation mode, the following changes are made by the installer:

- A new user/group named `flxrasvc` is created by the installer. No password is set for the account which puts it into a locked state. Therefore, it is not possible to log into the UNIX system using this account. This account name cannot be changed. Therefore, no changes should be made to this account to avoid breaking the installed agent.
- On Linux, if the Docker group exists, `flxrasvc` is added to the Docker group to allow `fnms-docker-monitor` to collect Docker inventory without root privileges
- The installer updates ownership on the installation directory to allow binaries to be run as the `flxrasvc` account.
- The installer updates ownership of the agent data directory to be owned by `flxrasvc`.
- The normal agent daemons (such as the usage agent `mgsusageag`, the schedule agent `ndtask`, and the Docker monitor `fnms-docker-monitor`) are configured to run as `flxrasvc`.
- A new entry to `/etc/managesoft.ini` is added to indicate that the agent is configured for least privileged operation; this setting is also propagated to the agent's `config.ini` main settings file.

The Flexera agent service account

The `flxrasvc` account is managed by the agent and the agent installer. No modifications should be made to this account.

Uninstalling an agent running in the least privilege operation mode will remove the `flxrasvc` account as well as the entire agent data directory from the UNIX system , because there is file system data owned by this account.

Additional processes launched by inventory collection

The agent uses `flxfsscan` and `flxoracleinv` to perform work on behalf of `ndtrack`, in both the full privilege default operation mode and the least privilege operation mode. It is an expected behaviour that these tools are launched several times while `ndtrack` is running.

Tip: *When you use trace to troubleshoot agent issues, it is recommended that the trace file name include* ***%p*** *to trace per-agent processes or* ***%n*** *to trace per-agent process names. This will help to separate different*

component tracing into logically separate trace files.

How to run agent components directly

Under normal operation, an agent configured for the least privilege operation mode will have the schedule agent daemon `ndtask` and the usage agent daemon `mgsusageag` running as the `flxrasvc` account. Any scheduled events, such as policy updates, inventory collection, or uploads, will be run by the scheduler under the `flxrasvc` account.

Important: *When running the agent components `ndtrack`, `mgspolicy`, and `ndupload` outside of the schedule (for example, manually through cron jobs), make sure to run these components as `flxrasvc`. Other user accounts on the machine will not be able to launch any agent binaries due to ownership and file permissions, and no user will be able to write to the agent data directory except `root`.*

Agent components can be run as `flxrasvc` through the following methods:

- Specify a user to impersonate with the `sudo` command.

```
sudo -u flxrasvc /opt/managesoft/bin/ndtrack -t machine
```

- Impersonate `flxrasvc` through `su`. Note that `su` must be run as `root`, otherwise the command will fail because the `flxrasvc` account is locked.

```
su - flxrasvc -c "/opt/managesoft/bin/ndtrack -t machine"
```

Agent Third-Party Deployment: Sample Sudoers File

```
       # Sample sudoers file for a _Linux_ machine, see the agent docs for details
on what tools are needed
       # by each supported Unix platform.
       #
       # Set up two aliases for OS level tools (FLEXERAOS) and Flexera provided
tools (FLEXERA)
       # Note that explicit paths to each tool is defined to narrow the tools that
can be launched
       # as root without a password through the flxrasvc account.
       #
       # Agents installed to custom install locations (where supported) need to
have these paths to
       # the Flexera tools updated to the custom install location.
       Cmnd_Alias FLEXERAOS = /usr/sbin/dmidecode
       Cmnd_Alias FLEXERA = /opt/managesoft/libexec/flxping, /opt/managesoft/
libexec/flxecmc, /opt/managesoft/libexec/flxsysinfo, /opt/managesoft/libexec/
flxps, /opt/managesoft/libexec/flxoracleinv, /opt/managesoft/libexec/flxfsscan

       # Allow the flxrasvc account to run the tools from the above aliases (and
only these tools) through sudo without requiring a password
       flxrasvc ALL=(ALL) NOPASSWD: FLEXERAOS, FLEXERA
```

Agent Third-Party Deployment: Enabling the HTTPS Protocol on UNIX Agents

The FlexNet inventory agent (or more precisely, its component executables) may make use of the HTTPS protocol for communications with inventory beacons. Whereas Windows systems can manage the security certificates for you, on UNIX and OS X some manual configuration is required.

Tip: *The options for checking certificates and checking certificate revocation are supported in networks using the IPv4 or IPv6 address families.*

Note: *The HTTPS protocol is only available to the installed agents, and is not available to the zero-footprint FlexNet Inventory Scanner.*

Remember: *Each inventory device may choose which inventory beacon it contacts at any given time, so that a decision to secure communications is normally a blanket decision covering your whole computing estate.*

There are three security levels which can be enabled for HTTPS, using two preference settings. From the highest level of security to the lowest, these are:

- Checking certificate(s) and excluding revoked certificates
- Checking certificate(s)
- Relying on encryption.

Checking certificate(s) and excluding revoked certificates

Authenticating HTTPS communications between the client (inventory device) and server (inventory beacon) may be achieved in either of two methods:

- Using unilateral (single-sided, or 'standard') TLS, where the client validates the server certificate(s)
- Using mutual TLS, where each side validates certificates offered by the other.

In both these cases, the client (inventory device) checks each HTTPS server certificate (from each inventory beacon with which it communicates). Checking the HTTPS server certificate involves having a local copy on the inventory device of all the public certificates (which may come from multiple certificate authorities (CA)) that are used to validate the HTTPS server certificates. These certificates must be available in the `/var/opt/managesoft/etc/ssl/cert.pem` file on the managed device (or an alternative folder — see Agent Third-Party Deployment: HTTPS CA Certificate File Format (UNIX) for more details). The device must also be able to download the certificate revocation list from an HTTP location, and/or perform an OCSP check for certificate revocation.

Only in the second case of mutual TLS, in addition to the above, when the `AddClientCertificateAndKey` preference is set, the managed inventory device provides the server (inventory beacon) with a client certificate that the server either accepts or rejects (for example, it may reject the certificate as being invalid, or expired).

Tip: *The inventory beacon does not save the client certificate locally, and does not require a chain of public certificates to authenticate the client certificate (and therefore it cannot check whether the client certificate has been revoked). This is a simpler check for both validity and the fact that the certificate has not expired. The server*

may be configured to:

- *Accept any valid, current client certificate that it is offered (but also allow HTTPS communications without a certificate)*
- *Ignore all client certificates (allowing HTTPS communications without any certificates)*
- *Require a valid, current client certificate before allowing any HTTPS communication with a client.*

For this level of security (using either form of TLS), both the `CheckServerCertificate` and `CheckCertificateRevocation` settings on the inventory device should be set to `True` (these are the default settings). When these are both true, a number of other settings can come into play, a few of which can be configured in the `mgsft_rollout_response` file that assists with deployment (see Agent Third-Party Deployment: Configuring the Bootstrap File for UNIX), and others must be modified in the `/var/opt/managesoft/etc/config.ini` file that functions in place of the Windows registry for UNIX-like platforms (see Agent Third-Party Deployment: Updating config.ini on a UNIX Device). The additional preferences are:

- AddClientCertificateAndKey (only applicable if you are using mutual TLS for authentication)
- PrioritizeRevocationChecks
- SSLCACertificateFile
- SSLCACertificatePath
- SSLCRLCacheLifetime
- SSLCRLPath
- SSLDirectory
- SSLOCSPCacheLifetime
- SSLOCSPPath.

Checking certificate(s)

This mid-level security model provides an encrypted channel and validation of either the HTTPS server (with standard TLS) or both the client and the server (with mutual TLS), but does not provide a way to check whether the certificate used to validate the server has been revoked. This may be adequate where you are confident of the longevity of your certificates, perhaps because you are using an internal certificate authority.

Checking the server certificate still requires that the CA certificate is installed on the target inventory device in the `/var/opt/managesoft/etc/ssl/cert.pem` file (and/or the alternative folder). If you are using mutual TLS, you have the `AddClientCertificateAndKey` preference set, and the inventory device still presents its certificate to the inventory beacon. As well, the `CheckServerCertificate` preference on the client must preserve its default value of `True`. Ignoring the revocation list is configured by disabling (setting to `False`) the `CheckCertificateRevocation` settings for all component agents on the managed device.

Tip: *It is also possible to generally disable for most agents, but create exceptions where a particular agent still checks for possible certificate revocation. For details, see CheckCertificateRevocation. (If you override the behavior for particular agent components, you may need to review the revocation settings listed above for the same components.)*

Relying on encryption

If you are confident of the security of your infrastructure, it is possible to ignore the server certificates entirely. This provides an encrypted channel of communication, but does not provide validation that the device is actually talking to the correct HTTPS server.

Disabling checking of the server certificate can be achieved by disabling (setting to `False`) both the `CheckServerCertificate` and `CheckCertificateRevocation` settings for all component agents on the managed device. In this mode of operation, the CA certificate is not required on the managed device.

Agent Third-Party Deployment: HTTPS CA Certificate File Format (UNIX)

By default, if the FlexNet inventory agent is (or in more detail, its component agents are) configured to use the HTTPS protocol to communicate with an inventory beacon, the certificate(s) for the HTTPS server are checked. This ensures that the agents are communicating with the correct inventory beacon server. Since the server certificate is authorized (signed) by a Certificate Authority, the process may include checking whether that Certificate Authority is trusted (that is, it may be an intermediate Certificate Authority that is itself trusted because it is authorized by a 'higher' Certificate Authority). The checking process continues until it reaches the root Certificate Authority (CA), one which the client device can recognize as trusted. This trust occurs when a CA certificate is 'known' to the client device.

Therefore certificate checking requires that a copy of the certificate for the root CA has been installed on the managed device. When a certificate being validated matches the locally-stored certificate for the root CA, trust is confirmed.

Tip: *Since an authorizing Certificate Authority may also revoke a previously-authorized server certificate that has somehow been compromised, by default the certificates are also checked for currency: that is, a check is made that each certificate in the chain (up to but excluding the root CA) has not been revoked. For the preference settings controlling revocation behaviors, see Agent Third-Party Deployment: Enabling the HTTPS Protocol on UNIX Agents.*

On Windows devices, the operating system handles all root CA certificates (and there may be many), and on Windows the FlexNet inventory agent uses those operating system services.

For UNIX-like platforms, the FlexNet inventory agent supports two ways of storing root CA certificates:

- Individual root CA certificates (in the PEM format discussed here) can be saved in the directory identified in the preference `SSLCACertificatePath` (default value `/var/opt/managesoft/etc/ssl/certs` — see SSLCACertificatePath). In this case, the files must be named with the CA subject name hash value (such as `9d66eef0`), with any duplicates differentiated by numeric file extensions (such as `9d66eef0.0`, `9d66eef0.1`, and so on).
- All the root CA certificates that are used by the HTTPS web servers supplying content to the managed device or receiving content from the managed device can be concatenated into a single file. (For many organizations, this will be a single certificate for a single root Certification Authority. It may even be a CA that is internal to the enterprise.) Each root CA certificate is added to the file named and saved as identified in the `SSLCACertificateFile` preference (default `/var/opt/managesoft/etc/ssl/cert.pem`), a simple naming convention. If you have multiple root CA certificates, simple shell commands allow the concatenation:

```
#!/bin/sh
rm cert.pem
```

```
for i in ca1.pem ca2.pem ca3.pem ; do
  openssl x509 -in $i -text >> cert.pem
done
```

Tip: *Before, between, and after the certificates in the concatenated file (that is, everywhere except between BEGIN and END tags), free text is allowed that can be used, for example, for descriptions of the certificates.*

This concatenated certificate file should be saved using the PEM format. Each PEM-format certificate should be base-64 encoded plain text surrounded by a `BEGIN CERTIFICATE` header and an `END CERTIFICATE` footer. That is:

```
-----BEGIN CERTIFICATE-----
MIIDiTCCAnGgAwIBAgIQWO/IibrLpZ5Hts3u3xH7TzANBgkqhkiG9w0BAQUFADAR
MQ8wDQYDVQQDEwZ0ZncyazMwHhcNMTAxMTI1MDEyMDM4WhcNMTUxMTI1MDEyODA1

......

wXvMSERKsNsJ6FwwXFGA3HBrRLTHzqzsfUlUAbV+SBm/FSFkuWsy4QWAuJCbnCnv
c3ClFHXqwaIq9UWvO5FR5kD4gK9LZOUY4B7tLTQmpJScFSiPZrIBa1cQ5uWl
-----END CERTIFICATE-----
```

To deploy the resulting certificate during deployment of FlexNet inventory agent to managed devices, see Agent Third-Party Deployment: Installing FlexNet inventory agent on UNIX.

Both storage methods may be used at once. The FlexNet inventory agent first checks the single concatenated file (where available), and then checks the folder of individual certificates. The checking stops at the first certificate that the managed device recognizes (that is, the first that matches the public certificate for the certificate authority that has been stored locally on the managed device).

Agent Third-Party Deployment: Updating config.ini on a UNIX Device

For OS X and UNIX managed devices, many preferences are stored in the `config.ini` file stored in `/var/opt/managesoft/etc`. This file acts as a “virtual registry”: that is, a repository on non-Windows platforms for settings that, on Windows platforms, are stored in the registry.

If you wish to update settings in `config.ini`, you can use the following approach, which can be completed manually or can be scripted with an appropriate deployment tool. In this description, the example used is to disable network sense in an installed FlexNet inventory agent.

To update the `config.ini` file:

1. Create a temporary file following the format of `config.ini`, but including only the section names containing changes, and the changed values.

   Example:

   ```
   [ManageSoft\Launcher\CurrentVersion]
   ```

```
NetworkSense=False
[ManageSoft\NetSelector\CurrentVersion]
SelectorAlgorithm=MgsSubnetMatch(,false)
```

2. Save (or deploy) this patch as a temporary file on the target device (where FlexNet inventory agent is locally installed), giving it a distinct name.

   For example, deploy (or save) the patch file to `/var/tmp/tempconfig.ini` on the target device(s).

3. Execute the `mgsconfig` tool with the `-i` option identifying your temporary patch file.

   Example:

   ```
   /opt/managesoft/bin/mgsconfig -i /var/tmp/tempconfig.ini
   ```

4. Remove the temporary file.

Inspecting the `/var/opt/managesoft/etc/config.ini` file shows that the settings in your temporary file have been applied to the `config.ini`.

Tip: *The `config.ini` file is used only for the FlexNet inventory agent installed locally on the target device (the Adopted case or Agent third-party deployment case). Do not attempt to use it for any of the other cases.*

Agent Third-Party Deployment: Uninstalling FlexNet inventory agent from UNIX

If you need to manually uninstall FlexNet inventory agent from a managed device, use the following command lines.

Note:

- *If the agent as been installed for the full privilege default operation mode, the uninstallation will leave behind the folder `/var/opt/managesoft` since it may contain a package cache that a future re-install of the can use.*
- *If the agent has been installed for the least privilege operation mode, the uninstallation will remove the `flxrasvc` account as well as the entire agent data directory from the UNIX system , because there is file system data owned by this account.*

Platform	Uninstallation command line

AIX	For removal from the standard installation path, use:
	`installp -u managesoft`
	If you have used a custom installation path, known as a User Specified Installation Location (USIL), you must include the same USIL so that FlexNet inventory agent is removed from the correct object repository (and keep the options in this order):
	`installp -R path -u managesoft`

Platform	Uninstallation command line
Debian Linux x86 and x86_64	`dpkg --purge managesoft`
HP-UX	`swremove managesoft`
Linux x86 and x86_64	`rpm --erase --verbose managesoft`
Mac OS X	`/opt/managesoft/bin/uninstall-managesoft.command -force`

Solaris x86 and SPARC	`echo action=nocheck > admin.mgs`
	`pkgrm -n -a admin.mgs ManageSoft`
	`rm -f admin.mgs`

Agent Third-Party Deployment: Protecting Your Customizations

As soon as its installation is complete, the FlexNet inventory agent attempts to contact the inventory beacon identified in the bootstrap configuration file, and if successful, requests its policy (including the schedule on which it should perform the specified actions).

You might desire either of two distinct outcomes here:

- Most likely, you may want this computing device to 'join the family', sharing the same schedule, policy, and controlled self-updating as all other instances of the FlexNet inventory agent on devices that were adopted automatically. If you take no special action, this is the default behavior.
- You may want to protect the unique settings on this instance of FlexNet inventory agent, such as a distinct set of failover inventory beacons. In this case, continue below.

To prevent standard policy over-writing your custom settings:

1. Prevent the distribution of policy to those devices that are outside all known targets:
   a. In the web interface for FlexNet Manager Suite, navigate to **Discovery & Inventory > Settings**.
   b. In the **Beacon settings** group, select the check box **Migration mode: Restrict inventory settings to targeted devices**.
   c. Click **Save**.

   This setting must remain permanently on for as long as you want to keep some of your devices (with FlexNet inventory agent installed) outside the reach of standard policies and settings (including the agent inventory schedule specified a little higher on the same product page).

Warning: *Turning on migration mode is a universal control that prevents every inventory beacon from answering requests from any random FlexNet inventory agents that call. When this control is set, inventory beacons provide policy and accept uploads only from FlexNet inventory agents installed on devices identified in targets that are in subnets assigned to each individual inventory beacon. You will therefore need to manage targets and assignments more carefully, and you may also wish to modify the preference SelectorAlgorithm.*

2. Ensure that the protected devices are never included in any target for discovery and inventory rules.

   These rules are specified in **Discovery & Inventory > Discovery and Inventory Rules**. There is no programmatic way to prevent these protected devices from being accidentally included in a target; it must simply be a matter of corporate guidelines that (for example) the specified subnet or the particular machine(s) must never be included in any target.

3. Plan to arrange your own updates to the FlexNet inventory agent on all protected devices, probably using the same configuration and deployment methods you have used for initial installation.

Agent Third-Party Deployment: Checking the Installed Version

You may need to verify the installed version of FlexNet inventory agent on a managed device, either for manual checking or as part of a scripted solution. You can inspect the version in the web interface for FlexNet Manager Suite; or you can dig deeper (including programmatically). As expected, the methods for digging deeper depend on the operating system on the managed device.

In the web interface

If the managed device has already reported inventory, you can inspect the results as follows:

1. Navigate to **Discovery & Inventory > All Inventory** (in the **Inventory** group).
2. On the **All Inventory** page, use searching or filters to find the managed device of interest, and click its **Name** to open its properties.
3. Select the **Applications** tab, and filter for `Inventory Manager Agent`. The installed version appears in the **Version** column.

Tip: *Ignore the `FlexNet Agent` listing, which is for a separate entity.*

The version shown is the friendly name (such as "2016 R2"). If you need to know major/minor version numbering, click through the **Product** value to reach the application properties, select the **Evidence** tab (with its default presentation for **Installer** evidence), and inspect the **Version** values for the evidence. Although typically wildcarded to match multiple builds, these show the major and minor numbering of the installer evidence that can produce the application record.

For managed devices on Microsoft Windows

Microsoft Windows Installer does not offer a simple command to list the version of an installed package.

However, you can check the registry. The installed version of the FlexNet inventory agent is stored in

```
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ManageSoft\ETCPVersion
```

(on 64-bit machines) or

```
HKEY_LOCAL_MACHINE\SOFTWARE\ManageSoft\ETCPVersion
```

(on 32-bit machines). This contains a string version showing major, minor, and build numbers (such as `11.04.19`).

For managed devices on UNIX and OS X systems

While these platforms do not have a registry, you can still inspect registry preferences in `/var/opt/managesoft/etc/config.ini`. (This is populated from the bootstrap file `/etc/managesoft.ini` during installation.) Look for the following section in `config.ini`:

```
[ManageSoft]

InstallDir=/opt/managesoft
ETCPInstallDir=/opt/managesoft
ETCPVersion=11.0.4
```

Note: *The format of the version number is different on UNIX-like systems than on Windows. Versions internally are numbered with four integers, a, b, c, and d. On Microsoft Windows, these are presented as* `a.bc.d` *(b and c are not separated) but on UNIX-like systems, they are presented as* `a.b.c` *(with the final integer unavailable). This variation means that the example given here for UNIX-like systems is the same release number as shown above for Microsoft Windows.*

This `config.ini` file remains available even after the FlexNet inventory agent is uninstalled (as noted in Agent Third-Party Deployment: Uninstalling FlexNet inventory agent from UNIX).

In addition, on UNIX-like systems, there are command lines available for checking the installed version of a package. You can use the following command lines on each of the platforms:

Platform	Version inspection command line
AIX	`lslpp -l managesoft.rte`
Debian Linux x86 and x86_64	`dpkg-query -W managesoft`
HP-UX	`swlist managesoft`
Mac OS X	`grep "ManageSoft " /Library/Receipts/ManageSoft.pkg/Contents/Info.plist`
RPM Linux x86 and x86_64	`rpm --query managesoft`

Platform	Version inspection command line
Solaris x86 and SPARC	`pkginfo -l ManageSoft`

Agent Third-Party Deployment: Troubleshooting Inventory

Inventory gathering and upload is a sophisticated chain from target inventory device through inventory beacon to central application server. For general trouble-shooting over the whole process, see the online help for FlexNet Manager Suite under *Inventory Beacons* > *Inventory Beacon Reference* > *Troubleshooting: Inventory Not Uploading*. This topic focuses entirely on inventory collection on the target inventory device.

After you have deployed and installed the FlexNet inventory agent, the regular log file for the `ndtrack` executable is identified in `[Registry]\ManageSoft\Tracker\CurrentVersion\LogFile` (see LogFile (inventory component)). In the Agent third-party deployment case, the default paths are:

- On Windows platforms, `$(TempDirectory)\ManageSoft\tracker.log`
- On UNIX-like platforms, `/var/opt/managesoft/log/tracker.log` (when the `ndtrack` executable runs as root).

For advanced trouble-shooting, you may require more advanced tracing and logging. You may also be asked to submit a trace file to assist the Support team at Flexera to solve difficult problems in your environment.

To configure advanced tracing for the installed FlexNet inventory agent:

1. In a flat text editor, open the `etcp.trace` file.

   In the Agent third-party deployment case, this file is co-located with the installed `ndtrack` executable on the target inventory device:

   - On Windows, the default is `C:\Program Files (x86)\ManageSoft\etcp.trace`
   - On UNIX-like platforms, the default is `/opt/managesoft/etcp.trace`.

2. Configure the name and location of the trace/log file that will be generated on the inventory device.

   The hash or pound character (#) identifies a comment. To "uncomment" a line in the `.trace` configuration file means to delete (only) the leading hash character. Choose one of the following lines, uncomment it, and optionally modify it to your requirements. On Windows:

   ```
   #filename=C:\ManageSoft.log
   #filename=C:\ManageSoft%p_%d_%t_%u.log        # filename pattern with everything!
   ```

   On UNIX-like platforms:

   ```
   #filename=/tmp/log/mgstrace.log
   #filename=/tmp/log/ManageSoft%p_%d_%t_%u.log # filename pattern with everything!
   ```

   See the notes within the file header for the use of the supported variables within the file name.

Tip: *It is best practice to use a pattern that includes (at least) either a date stamp (%d) or a sequential number (%u). Without these, the fixed file name means tracing information is appended to the same file with every inventory collection. This can quickly produce a trace file too large for text editors to read, and too hard to manage in terms of disk space. Variables in the file name trigger creation of a new file each time the associated variable is changed (or, for %u, at every invocation of* `ndtrack`*).*

Important: *The log file path:*

- *Must be on the same drive as the* `ndtrack` *executable (on Windows devices)*
- *Must exist and be writable before the* `ndtrack` *executable is next invoked (tracing does not create any directories, and does not function if any directory in the specified path is missing or unwritable).*

3. Uncomment the lines for which you want to enable tracing (ensuring that the uncommented line now starts with a plus sign).

   The tracing controls are arranged hierarchically. For example, uncommenting the one line for `+Inventory/Tracker` enables tracing for all child controls, as in this example:

```
+Inventory/Tracker
#+Inventory/Tracker/Preferences
#+Inventory/Tracker/Environment
#+Inventory/Tracker/Hardware
#+Inventory/Tracker/Hardware/WMI
#+Inventory/Tracker/Hardware/WMI/Class
#+Inventory/Tracker/Hardware/WMI/Instance
#+Inventory/Tracker/Hardware/WMI/Property
#+Inventory/Tracker/Hardware/Processor
#+Inventory/Tracker/Hardware/Memory
#+Inventory/Tracker/Hardware/Hypervisor
#+Inventory/Tracker/Hardware/DiskDrive
#+Inventory/Tracker/Registry
#+Inventory/Tracker/Registry/Keys
#+Inventory/Tracker/Registry/Values
#+Inventory/Tracker/Package
#+Inventory/Tracker/Package/Info
#+Inventory/Tracker/Software
#+Inventory/Tracker/Software/Directory
#+Inventory/Tracker/Software/File
#+Inventory/Tracker/Software/Version
#+Inventory/Tracker/Oracle
#+Inventory/Tracker/Oracle/Listener
#+Inventory/Tracker/Generate
#+Inventory/Tracker/Compress
#+Inventory/Tracker/Upload
```

   This setting enables (almost) *all* tracing related to the tracker component (`ndtrack`). You can also create an exemption to the general setting by making the appropriate line starts with a minus sign. For example, this one-line change within the above:

```
-Inventory/Tracker/Registry
#+Inventory/Tracker/Registry/Keys
#+Inventory/Tracker/Registry/Values
```

turns off tracing for everything related to gathering inventory from the Windows registry (that is, the disable setting is also inherited by the `Inventory/Tracker/Registry/Keys` and `/Values` controls).
One control that may affect the tracker component is outside this set. Because the tracker attempts an upload to the inventory beacon as soon as inventory gathering is complete, its tracing is affected by the `+Communication/Network` setting (along with the `ndupload` and `ndlaunch` components). This enables tracing of all network communications, including server certificate checking and the like.

Some common choices for tracing the inventory gathering process are listed in the table below.

4. To turn off tracing for an individual line that has previously been enabled, either comment out the line again, or switch the plus sign to a minus sign at the start of the line. A quick way to turn off tracing but keep all the settings for future use is to comment out only the `filename` setting that specifies the log file.

Some of the more commonly used tracing options for the tracker include the following:

Category	Option	Notes
Networking	`+Communication/Network`	Traces all low-level upload and download actions (whether HTTP or HTTPS). It includes HTTPS certificate checking and related areas. Covers actions by the `ndtrack`, `ndupload`, and `ndlaunch` components.
All inventory	`+Inventory`	Traces all inventory operations, which on large inventory tasks, could result in a sizable trace file.
All tracker	`+Inventory/Tracker`	This traces almost all operations of the `ndtrack` executable.
Preferences	`+Inventory/Tracker/Environment`	Shows active preferences, whether set in the registry (on Windows platforms) or in the `config.ini` file (on UNIX-like platforms), or inbuilt default values.
Hardware inventory	`+Inventory/Tracker/Hardware`	Traces all hardware inventory classes visible in the `<Hardware>` node in the `.ndi` inventory file, including CPU information and virtualization.
Software inventory	`+Inventory/Tracker/Package`	Traces the inventory operations that populate the `<Package>` nodes in the `.ndi` inventory file.
Software inventory	`+Inventory/Tracker/Software`	Tracing mainly for file inventory gathering (such as the `<Content>` and `MD5` nodes of the `.ndi` file).
Oracle inventory	`+Inventory/Tracker/Oracle`	When the inventory device hosts Oracle Database, this is the tracing for local Oracle inventory.
Operations	`+Inventory/Tracker/Generate`	Traces the preparation of the `.ndi` inventory file(s), keeping in mind that on an Oracle Database server, there may be multiple files generated.

Category	Option	Notes
Operations	`+Inventory/Tracker/Compress`	Traces the compression of the `.ndi` file into a `.gz` archive.
Operations	`+Inventory/Tracker/Upload`	Traces the upload of the `.ndi.gz` archive to an inventory beacon by the tracker. ***Tip:*** *If the immediate upload by the tracker fails for some temporary reason, the upload is attempted again later by the `ndupload` component. While this component does not provide this same level of operations tracing as the tracker does, you can enable `+Communication/Network` for low-level tracing of each step in the upload interaction.*

4

Zero-Footprint: Details

This chapter provides great detail about the FlexNet inventory core components when these are included as part of the standard installation of the FlexNet Beacon code on an inventory beacon. When used in this environment, the functionality of the FlexNet inventory core components is augmented by other code elements that are part of FlexNet Beacon, making this use case unique.

In this configuration on an inventory beacon, the FlexNet inventory core components are capable of collecting hardware and software inventory from separate target devices, using a remote execution technique fully described in this chapter.

Although the inventory component (`ndtrack`) executes in the context of the target inventory device, it is removed after each inventory exercise. Since nothing is permanently installed on the target inventory device, this is called the Zero-footprint case. For details of the distinct use cases, refer back to Understanding What, Where, How, and Why.

This document provides a consistent set of data (as far as possible) across all the different use cases, each in its own chapter. This means that, once you have chosen your preferred use case, you can focus only on the details for that one, and ignore all other use case chapters.

In addition to the distinct chapters for the different use cases, you should also review the subsequent chapter on functionality that is common throughout. This is followed by detailed reference material on command lines, preferences, file formats, and the like.

Zero-Footprint: Normal Operation

Because of the requirement that no executables are permanently installed on target inventory devices in the Zero-footprint case, there are some variations across platforms in how the remote execution operates. However, the principles of the process are consistent. This description assumes that you have configured the system to allow for Zero-footprint discovery and inventory collection (described in Zero-Footprint: Implementation). The entire operational process is covered, including what happens to the collected data after it uploaded to the inventory beacon and beyond. Each numbered step provides a summary point, followed by further specific details that you can skip over until needed.

Important: *This method of discovery and inventory gathering is not currently supported in IPv6 networks.*

The process always begins with discovery, and moves directly to inventory gathering from the appropriate devices.

1. On the schedule declared an the applicable rule, the FlexNet Beacon engine conducts a sweep of its assigned

subnet.

This sweep may use either (or both) of:

- A network scan, with testing of a specified list of ports (this method is mandatory for discovery and inventory of UNIX-like machines)
- A scan by the Windows Computer Browser service.

To reivew or update your choice, navigate to **Discovery & Inventory > Discovery and Inventory Rules > Actions**, click the pencil icon to edit an action, and then go to the **Discovery of devices** section.

A set of IP addresses is returned.

2. Each IP address is assessed as follows:

   a. Is the IP address within a subnet that is assigned to this inventory beacon?

      If not, the IP address is discarded, and the next one is processed. Log files for the work of the FlexNet Beacon engine are saved in *%CommonAppData%*\Flexera Software\Compliance\Logging\BeaconEngine.

   b. Is this IP address matched by the target in the rule (for which the action includes discovery and inventory gathering) that has been triggered?

      If not, the IP address is discarded.

   c. Appropriate ports are probed.

      There may be several reasons why ports are probed:

      - If the action settings included **Discover devices using network scan**, there is a list of default ports, including for example port 22 (for SSH on UNIX-like platforms) and port 135 (for NetBIOS on Windows), and others. You may have added additional ports to this set.
      - Additional parts of the action definition, such as **Oracle VM discovery and inventory**, **Microsoft SQL Server discovery and inventory**, **VMware discovery and inventory**, and so on may also specify ports, on which specialized probes are conducted to identify the services you are checking for.
      - If you have selected **TNS names file** as a discovery method in the **Oracle discovery and inventory** section of the action specification, and there is a `tnsnames.ora` file present in *%CommonAppData%*\Flexera Software\Repository\TNSNames\ on the inventory beacon, the servers (and ports) identified in that file are also probed as part of discovery. (If there are multiple `.ora` files, each is processed in turn.)

   d. Are there credentials for this device recorded in the local Password Manager on the inventory beacon?

      This is confirmed by a successful log in. If not, the IP address is discarded.

   e. With successful login, a query is used to identify the operating system on the device (so that appropriate command lines can be constructed), and a check is made for the presence of the FlexNet inventory agent.

      - For Windows devices, the status of the `ndinit` service is checked. If it is present, this also gives the installed version of the FlexNet inventory agent, and an indication of its healthy operation.
      - For UNIX-like devices, the following command is used both to identify the operating system type and to determine whether the device is already adopted (such that the FlexNet inventory agent is already

installed on the device, as shown by the `ETCPVersion` result):

```
/bin/sh -c "PATH=$PATH:/bin:/usr/bin;
echo \"UnameKernel=`uname`\";
[ -r /etc/managesoft.ini ] &&
grep ETCPVersion /etc/managesoft.ini || echo \"ETCPVersion=NONE\"”
```

If found by these means, the FlexNet inventory agent is logged in the inventory database as installed on the discovered device.

Note: *The discovery of the installed FlexNet inventory agent by this means does not have the impacts that you may imagine:*

- *It does not cause its appearance in the web interface for FlexNet Manager Suite, and in particular does not drive the result shown in* ***Discovery & Inventory > All Discovered Devices*** *in the* ***Agent installed*** *column. (This column is derived from inventory results, not from discovery results, allowing for results from third-party inventory tools to also be reflected in the column.)*
- *It does not influence the decision about whether to apply any Zero-footprint inventory gathering specified in the current rule to the device. When specified, this kind of inventory gathering goes ahead on all devices except those for which their policy specifies adoption. As a consequence, if you have used third-party tools (or manual effort) to deploy the full FlexNet inventory agent to this device, and the device is outside the policy for adoption but inside the target(s) for a rule specifying Zero-footprint inventory gathering, then you collect inventory from both methods: Zero-footprint inventory gathering, and inventory taken by the locally-installed FlexNet inventory agent. While uncommon, it is then possible that your customized settings could cause resulting data to flip-flop based on which inventory method was used most recently.*

The main purpose of this discovery check for an installed FlexNet inventory agent comes in the next step, in the adoption check.

This completes discovery, and any device still under consideration is included in the list of discovered devices visible in the web interface of FlexNet Manager Suite. Meanwhile, if the rule included any inventory gathering as part of its action, the process continues.

3. If, in the **General hardware and software inventory** section of the action settings for the relevant rule, the **Gather hardware and software inventory** check box is selected, the FlexNet Beacon engine checks the `BeaconPolicy.xml` file to see whether the current device is listed (in the net result of all targets) for adoption.

   - If the device policy is for adoption, two consequences follow:

     a. The discovery result for an installed FlexNet inventory agent is assessed. If the FlexNet inventory agent is not present, adoption is initiated immediately.

     b. When the FlexNet inventory agent is present (or adoption has been initiated in this pass), the Zero-footprint inventory process is terminated for this device, and the process moves onto the next device in the list. (If this device was targeted for Microsoft SQL Server or Microsoft Hyper-V inventory collection, these forms of Zero-footprint inventory collection are also terminated in this case, and are left to the installed FlexNet inventory agent.)

   - If the device is not targeted for adoption (or specifically excluded from adoption in at least one target), the process continues.

> ***Tip:*** *The adoption test is entirely based on target policy settings in the web interface. This means that, if you deployed the FlexNet inventory agent independently (the Agent third-party deployment case) and also include the device in a target for inventory gathering in the Zero-footprint case, inventory gathering proceeds twice for this device.*

4. The method of gathering general hardware and software inventory varies across platforms:

   Microsoft Windows:

   a. The FlexNet Beacon engine, which is still logged into the target device, creates and runs a Windows service (with the display name `mgs-GUID`, executing `mgsreservice.exe`).

   b. The service executes the `ndtrack.exe` inventory component from the inventory beacon file share `mgsRET$`.

      The executables are available on the inventory beacon in the default path `%ProgramFiles%\Flexera Software\Inventory Beacon\RemoteExecution\Public\Inventory` (defined in the Windows share `mgsRET$`) .

   c. Because the command line parameters passed to `ndtrack` included the `-o UploadLocation` preference that identified the parent inventory beacon, the `ndtrack` component immediately attempts an upload of the collected inventory.

      On the inventory beacon, uploaded FlexNet inventory files are saved in *%CommonAppData%*`\Flexera Software\Incoming\Inventories`.

      > ***Tip:*** *Since only one inventory beacon is identified in the preference, there is no fail-over should the specified inventory beacon be unavailable for any reason, including that it requires credentials not known to `ndtrack`. (Fail-over inventory beacons are identified only for installed FlexNet inventory agents that are self-managing based on collected device policy, in either the Adopted case or the Agent third-party deployment case.)*

   d. The service is closed down.

   The following artifacts remain on the target inventory device, and are over-written on the next execution of the same process:

   - An uncompressed inventory (`.ndi`) file is left in `%ProgramData%\ManageSoft Corp\ManageSoft\Tracker\ZeroTouch`
   - A `tracker.log` log file is saved on the target inventory device in `C:\Windows\temp\ManageSoft`.

   UNIX-like platforms

   a. The FlexNet Beacon engine, which is still logged into the target device, executes `sudo` to elevate its privileges to `root` level.

      > ***Tip:*** *It is technically possible to run Zero-footprint inventory collection from UNIX-like devices as a non-root user; but this prevents collection of complete inventory (for details, see Zero-Footprint: Non-Root Accounts).*

   b. The FlexNet Beacon engine then executes `ssh` to establish a secure link.

c. The engine then uses `scp` to copy the FlexNet inventory core components to the target device.

For convenience in dealing with the variety of target platforms, these are deployed in the form of ndtrack.sh, a 'self-installing' script. This is available on the inventory beacon in the default path `%ProgramFiles%\Flexera Software\Inventory Beacon\RemoteExecution\Public\Inventory`.

Tip: *The same directory may also contain `ndtrack.ini`, a configuration file (for UNIX-like machines only, and only in the Zero-footprint or FlexNet Inventory Scanner cases) that may contain preferences applicable to `ndtrack`. With the restriction that preferences apply only to this one component, this file has the same schema as `config.ini`, the substitute registry for agent preferences for the cases where the FlexNet inventory agent is locally installed on the target device (see, for example, Agent Third-Party Deployment: Updating config.ini on a UNIX Device). However, there are no fail-over settings included in `ndtrack.ini`, for the reasons noted below. It is not mandatory to supply this file, since `ndtrack.sh` has built-in default settings that provide all necessary values beside those supplied in the command line. However, if you wish to override any of these default values, you can customize the `ndtrack.ini` file available in this same folder as `ndtrack.sh`.*

d. The ndtrack.sh script is executed, identifies the particular platform, and unzips the appropriate executable either into the home directory of the `root` account or (if that home directory is `/`) into `/var/tmp`.

e. The `ndtrack` component collects the inventory.

f. Because the command line parameters passed to `ndtrack` included the `-o UploadLocation` preference that identified the parent inventory beacon, the `ndtrack` component immediately attempts an upload of the collected inventory.

On the inventory beacon, uploaded FlexNet inventory files are saved in *`%CommonAppData%`*`\Flexera Software\Incoming\Inventories`.

Tip: *Since only one inventory beacon is identified in the preference, there is no fail-over should the specified inventory beacon be unavailable for any reason, including that it requires credentials not known to `ndtrack`. (Fail-over inventory beacons are identified only for installed FlexNet inventory agents that are self-managing based on collected device policy, in either the Adopted case or the Agent third-party deployment case.)*

g. The FlexNet Beacon engine, still running as `root`, deletes the executable, deletes the `ndtrack.sh` script, and logs out.

The following artifacts remain on the target inventory device, and are over-written on the next execution of the same process:

- An uncompressed inventory (`.ndi`) file is leftwhen inventory collection is (as usual) run as `root`, in `/var/tmp/flexera/tracker`; or if inventory collection is run as another user, in `/var/tmp/flexera.`*`userName`*`/tracker`
- Log files are saved on the target inventory device when inventory collection is (as usual) run as `root`, in `/var/tmp/flexera/log`; or if inventory collection is run as another user, in `/var/tmp/flexera.`*`userName`*`/log`. One called `ndtrack.log` is created by the shell script wrapper, and `tracker.log` is the logging output from the `ndtrack` component itself.

5. In parallel with the above process for general hardware and software inventory, the FlexNet Beacon engine also conducts specialized inventory gathering on discovered devices matching the specifications in the action for the rule being executed.

   These are the devices for which inventory gathering has been specified for VMware, Microsoft Hyper-V, Citrix Virtual Desktops, Oracle Database environments or Oracle VM infrastructure. Each type of inventory gathering creates its own `.ndi` inventory file. These are added to the uploaded general inventory files in *%CommonAppData%*`\Flexera Software\Incoming\Inventories`.

6. FlexNet Beacon (the code entity on the inventory beacon) uploads the inventory data to its parent on a schedule set by the Microsoft Scheduled Task `Upload FlexNet logs and inventories` (by default, repeating every minute throughout the day).

   The checking cycle when the folder is empty is very quick and does not perceptibly load the inventory beacon, even though it is frequently repeated. The parent of an inventory beacon may be the central application server, or another inventory beacon if these have been arranged in a hierarchy. In the latter case, each inventory beacon in turn repeats the upload process until the data reaches the application server.

7. On the application server (or, in a scaled-up system with separate servers, the inventory server), the web service `ManageSoftRL` receives the uploaded inventory file(s).

   These are processed immediately, being loaded into the internal inventory database. If the service gets overloaded, it will temporarily spool incoming files to its local *%CommonAppData%*`\Flexera Software\Incoming\Inventories` directory. From here, file import is resumed under the control of the Microsoft scheduled task `Import inventories`, which is triggered every 10 minutes.

8. On the next inventory import and license consumption calculation, the inventory data is collected from the inventory database, socialized as necessary, and imported into the compliance database. Here it is used in license calculations, and made available in management views and reports.

   This import step can be triggered in one of three ways:

   - Normally, the batch scheduler triggers an import daily (by default, at 2am local time on your application server), with the license consumption calculation triggered thereafter. This default time is configurable by editing the Microsoft scheduled task `Inventory import and license reconcile` on your application server (or, in larger implementations, batch server).
   - An operator in the `Administrator` role can choose to import the waiting inventory and trigger license consumption calculation, or reconciliation, as soon as possible (navigate to **License Compliance > Reconcile**).
   - For testing, a knowledgeable system administrator could use a command line on your application server (or, in a scaled-up system, your batch server) like:

     ```
     BatchProcessTask.exe run InventoryImport
     ```

     (for details, see the *Server Scheduling* chapter in *FlexNet Manager Suite System Reference*).

Zero-Footprint: Non-Root Accounts

By default for UNIX-like target devices, the FlexNet Beacon engine collects Zero-footprint inventory by escalating its privileges to act as `root` on the target device.

It is technically possible to collect inventory as a non-root user (that is, the ndtrack executable will run and produce a .ndi document). However, information that cannot obtained without root or administrative rights includes:

Platform	Missing inventory details (non-root users)
Linux	• BIOS details (dmidecode): serial number, UUID, manufacturer, model, chassis type • All hard disk information (from device files).
Solaris	• MAC addresses of network adapters • x86 BIOS details (dmidecode): model, manufacturer • SPARC model using OpenPROM interface. It fails over to using the sysinfo SI_PLATFORM value which can be different.
HP-UX	• SD-UX installation evidence from swlist if access has been locked down with swreg or swacl • vPar evidence including VMType, VMName and vPar capacity (vparstatus requires root) • Hard disk drive properties including capacity.
Mac OS X	Mac OS X package bundle paths under /Applications or /System/Library that are not accessible by the executing user.
All UNIX-like platforms	• Collection of inventory for IBM Db2 Database (and optional add-ons) is blocked for non-root accounts • Collection of inventory for IBM MQ (previously WebSphere MQ) is blocked for non-root accounts • Collection of Oracle inventory is blocked for non-root accounts • File evidence from any file system path not accessible by the executing user • InstallAnywhere, InstallShield Multiplatform, Oracle Universal Installer evidence under paths not accessible by the executing user.

If this reduced functionality is acceptable for certain inventory targets, you can configure the account for these devices in the Password Manager to prevent elevation of privileges.

Zero-Footprint: System Requirements

These details apply to the use of the FlexNet inventory core components installed as standard on each inventory beacon server (as part of the installation of FlexNet Beacon). In this configuration, the FlexNet inventory core components use remote execution techniques to gather inventory from separate targets (the techniques are naturally

different for Windows and UNIX-like platforms, and are detailed in Zero-Footprint: Normal Operation). Therefore, there are two sets of requirements that become important in this Zero-footprint case:

- The requirements on the inventory beacon server itself
- Any impacts on the target inventory devices.

Note: *At the risk of confusion, we should note that an inventory beacon can also be a managed device, a computer for which you want to collect its own inventory. This uses a separate code collection. For the remainder of this topic, we are overlooking this aspect, and focusing exclusively on the capacity of the FlexNet Beacon code installed on the inventory beacon to remotely collect inventory from other target inventory devices, excluding itself.*

Supported platforms

On the inventory beacon itself, the FlexNet inventory core components impose no limits on the supported platforms. The FlexNet Beacon code (including the FlexNet inventory core components) can be installed on the following platforms:

- Windows Server 2012, 2012 R2, 2016, 2019, 2022
- Windows 8, 10, 11.

For remote inventory collection in the Zero-footprint case, inventory can be gathered from the following platforms:

Microsoft Windows	UNIX-like platforms

• Windows Server 2008 R2 x64, 2012, 2012 R2, 2012 R2 SP1, 2016, 2019, 2022
• Windows Server Core 2008, 2008 R2 x64, 2012, 2012 R2
• Windows Server Standard (previously known as Windows Server Core) 2016, 2019
• Windows Vista, 7, 8, 10, 11
 	
Note: *Windows systems that can run on ARM-based devices are also supported.*
	• AIX 7.1 LPARs with Technology Level 5 or later, AIX 7.2
	• Amazon Linux 2, 2023 (ARM64/AArch64; x86, 32-bit and 64-bit)
	• CentOS 6-7.5 (x86, 32-bit and 64-bit); 7.6-9 (x86 64-bit only)
	• Debian Linux 8–11.3 (x86, 32-bit and 64-bit); 11.5, 12.0 (x86 64-bit only)
	
	Note: *For Debian Linux 9.4, 9.5 and 10 (both 32-bit and 64-bit), minimal installations of the OS core omit the `ifconfig` command. This prevents collection of the IP address and MAC address in inventory. The `root` user can use the following command to restore this functionality:*
	`apt-get install net-tools -y`
	• Fedora Linux 26 (x86, 32-bit and 64-bit); 27-38 (x86 64-bit only)
	• HP-UX 11i v3 (Itanium only), vPars/nPars
	• macOS 10.15.4–14 (applicable for both Intel and Apple M-series processors)
	
	Note: *If you need to run the FlexNet inventory agent of version 20.1 or earlier on an Apple M-series processor ("Apple silicon"), Rosetta 2 must be installed and running. This is Apple's solution for transitioning most Intel-based applications to run on Apple silicon. There are two possible command formats for installing Rosetta 2:*
	◦ *Interactive installation that asks for agreement to the Rosetta 2 license:*
	`/usr/sbin/softwareupdate --install-rosetta`
	◦ *Non-interactive installation:*
	`/usr/sbin/softwareupdate --install-rosetta --agree-to-license`

Microsoft Windows	UNIX-like platforms
	• Nutanix AHV hypervisor ***Note:*** *All versions that are within their product support lifecycle and use the* `Libvirt` *library are supported.* • OpenSUSE Leap 42.2, 42.3 (x86, 32-bit and 64-bit); 15-15.4 (x86 64-bit only) • Oracle Linux 6.0–6.10 (x86, 32-bit and 64-bit); 7.0-8.7 and 9.0-9.1 (x86 64-bit only) • Photon OS 3.0-5.0 • Red Hat Enterprise Linux (RHEL) 6.0-6.10 (x86, 32-bit and 64-bit); 7.0-8.8 and 9.0-9.2 (x86 64-bit only) • Rocky Linux 8, 9 • Solaris 10–11 (SPARC), Zones for versions 10–11 • Solaris 10–11.4 (x86), Zones for versions 10–11 • SUSE Linux Enterprise Server 12 SP3, 12 SP4, 12 SP5, 15, 15 SP1, 15.2-15.5 (x86 64-bit only) • Ubuntu 14–17.04 (x86, 32-bit and 64-bit); 17.10-23.04 (x86 64-bit only) Linux on IBM zSystems platforms are also supported. The following Linux distributions are certified for usage on the IBM zSystems architecture. For details about supported versions on different hardware types, see https://www.ibm.com/support/pages/linux-ibm-z-tested-platforms. • Red Hat Enterprise Linux • SUSE Linux Enterprise Server • Ubuntu

Disk space requirements

On the inventory beacon itself, the FlexNet inventory core components are included in the disk space requirement for FlexNet Beacon:

- 1 GB of free disk space for each 10,000 devices from which FlexNet inventory is collected.

On target inventory devices (in the Zero-footprint process), the following are platform-specific disk space requirements:

- Windows: Where the target device is a typical workstation, in the order of 2MB; and for an Oracle server, in the order

of 5MB.

- UNIX-like platforms: The downloaded code is approaching 23MB, and it then looks for an additional 16MB of working disk space in either (in priority order):

  1. The home directory of the account running the inventory (unless that directory is `/`)
  2. `/var/tmp`.

  Where this space is not available, inventory collection is not attempted.

After Zero-footprint inventory collection, the following (non-executable) files are left on the target inventory device as a potential aid to process validation or trouble-shooting, and are all replaced at the next iteration of the same process:

- An uncompressed inventory (`.ndi`) file is left:
  - For Windows devices, in `%ProgramData%\ManageSoft Corp\ManageSoft\Tracker\ZeroTouch`
  - For UNIX-like devices, when inventory collection is (as usual) run as `root`, in `/var/tmp/flexera/tracker`; or if inventory collection is run as another user, in `/var/tmp/flexera.`*`userName`*`/tracker`.
- Log files are saved on the target inventory device:
  - For Windows devices, in `C:\Windows\temp\ManageSoft`
  - For UNIX-like devices, when inventory collection is (as usual) run as `root`, in `/var/tmp/flexera/log`; or if inventory collection is run as another user, in `/var/tmp/flexera.`*`userName`*`/log`.

The following log file is available on the target inventory device in the Zero-footprint case:

- `tracker.log` — Generated by the inventory component, `ndtrack`

Memory requirements

On the inventory beacon itself, the memory demands in the Zero-footprint inventory collection process are negligible, and entirely covered by the standard specification for the inventory beacon of 1GB minimum RAM, 2GB or higher recommended.

On target inventory devices, the memory requirements are:

- Minimum RAM: 512 MB
- Recommended RAM: 2 GB

In general, through a cycle of inventory gathering and upload, the memory demand is in the order of 5-30 MB.

Communication protocols and ports

All ports used in Zero-footprint discovery and inventory collection are configurable to any value either through preference settings or by including the port number in URL settings.

The default ports used for discovery depend on what the inventory beacon is tasked to discover:

- ICMP is used for ping discovery of networked computers
- 135 (default) on the target device for WMI-based discovery of Hyper-V or XenDesktop (other ports depend on your configuration of WMI)

- 80 (with HTTP), 443 (with HTTPS) for VMware ESX/VCenter (default values, and of course using TCP)
- 1433 for Microsoft SQL Server (default)
- 1521, 2483 for Oracle DB (default)
- 137 for NetBIOS discovery of networked computers (default)
- 161 for SNMP discovery of networked computers (default)

The default ports for communications required for the Zero-footprint inventory case are:

- Windows: Outbound from inventory beacon to set up the service: for SMB, port 445; and for NetBIOS, port 139
- UNIX-like platforms: Outbound from inventory beacon to establish secure connection and copy the FlexNet inventory core components (in self-installing form): port 22
- Additional ports outbound from inventory beacon to an Oracle Database: ports 1521 and 2483
- Additional ports outbound from inventory beacon to a virtual machine under either ESX or VCenter: 80 and 443
- Windows and UNIX: Inbound from FlexNet inventory core components on target device: File upload using HTTP protocol: port 80
- Windows only: Inbound from FlexNet inventory core components on target device: File upload using HTTPS protocol: port 443

Tip: *File upload using HTTPS protocol is not directly supported for UNIX-like platforms in the Zero-footprint case. However, the standalone inventory agent* `ndtrack.sh` *for UNIX-like platforms supports inventory file upload using HTTPS protocol as long as SSL certificate is provided and the* ***SSLDirectory*** *option is specified on the command line.*
For example, the following command uses the ***SSLDirectory*** *option to pass the path of the SSL certificate that is used on the UNIX-like platform to* `ndtrack.sh` *and uses the* ***UploadLocation*** *option to collect and upload inventory to the inventory beacon.*

```
ndtrack.sh -o UploadLocation="https://InventoryBeacon:port/ManageSoftRL" -o
SSLDirectory="/mydir/certs/ssl"
```

For more information about using `ndtrack.sh`*, see ndtrack Command Line.*

- Additional ports may be required if supporting a proxy.

Supported packages to inventory

FlexNet inventory can include data from most package technologies supported by the operating systems, and some additional third-party packaging technologies:

Platform	Supported package technologies
All platforms	InstallAnywhere (IA), InstallShield Multiplatform (ISMP), BEA/Oracle Installer (BEA), Oracle Universal Installer (OUI), IBM Installation Manager (IIM)
AIX	LPP, RPM

Platform	Supported package technologies
HP-UX	Software Distributor SD-UX Package
Linux	RPM (Red Hat, CentOS, Oracle, SuSE, Fedora, etc), DPKG (Debian, Ubuntu).
macOS	Mac Application Bundle, Mac Package Bundle
Solaris	Sys V Package (pkg), IPS
Windows	MSI, Add/Remove Programs Registry Key

However, FlexNet inventory cannot collect data from some of the less common or newer operating system technologies and many third-party technologies. Some known examples include:

- All platforms — IBM InstallStream, IBM Tivoli Netcool Installer
- macOS — Mac flat package.

System load benchmarks

The following notes reflect observed behavior on sample target inventory devices during collection of FlexNet inventory:

Task	Run duration (seconds)	CPU usage (seconds)	CPU usage (% of single core)	Memory usage	Network load
Inventory collection	13 to 240 s	5 to 130 s	10% to 50%	4 MB to 20 MB	10 KB to 200 KB per upload

Zero-Footprint: Accounts and Privileges

In the Zero-footprint case, when the FlexNet inventory core components (installed as part of the FlexNet Beacon code base) reach out to gather hardware and software inventory from remote target inventory devices, there are accounts required on both the local inventory beacon and on the remote target device.

On the inventory beacon, no *separate* account or privileges are required for the inventory beacon to exercise the Zero-footprint case: the FlexNet Beacon itself must be executed by a service account able to log in as a batch job, and to run scheduled tasks (and, if the inventory beacon is running IIS, to run IIS application pools). This same account executes the remote discovery and inventory collection tasks.

Tip: *A service account configured for the above privileges does not normally allow interactive login. To access the FlexNet Beacon interface on an inventory beacon requires a separate account with local administrator privileges.*

One of the first actions when Zero-footprint inventory gathering is triggered is to configure software on the target inventory device to complete the inventory gathering action (the methods vary across platforms, and are detailed in Zero-Footprint: Normal Operation. This means that there may be two accounts required for each device:

- The initializing account
- The operational account that actually gathers the inventory.

Naturally, the requirements vary across platforms.

On Microsoft Windows target devices

- The initializing account:
  - May be either a Windows domain account, or a local account on the target device
  - Requires full access to the Windows Service Control Manager on the target device (specifically, it must have the `SC_MANAGER_ALL_ACCESS` access right)
  - Must be appropriately registered in the secure Password Manager on the inventory beacon that is responsible for collecting inventory from this target device (for details, see *FlexNet Manager Suite Help* > *Inventory Beacons* > *Password Management Page* and its child topics)
  - May conveniently be the `LocalSystem` account, since this is required for the following operational stage.
- For the operational account, FlexNet inventory core components (and in particular the `ndtrack` component) run as the `LocalSystem` account.

On UNIX-like target devices

- The initializing account:
  - Is a local account on the target inventory device
  - Has `ssh` privileges on that device
  - Must be appropriately registered in the secure Password Manager on the inventory beacon that is responsible for collecting inventory from this target device (for details, see *FlexNet Manager Suite Help* > *Inventory Beacons* > *Password Management Page* and its child topics)

  Note: *When you save the SSH account details in the Password Manager, be sure to specify the additional details for elevation of account privileges with your preferred tool (such as `sudo` or `priv`).*
- For the operational account, FlexNet inventory core components (and in particular the `ndtrack` component) run as `root`.

  Tip: *As always, it makes no difference whether you invoke the tracker directly as `root`, or whether you run as another account and use `sudo` (or similar) to elevate to `root` before invoking the tracker.*

Zero-Footprint: Implementation

The Zero-footprint case does not require any software deployment, since the FlexNet inventory core components are installed on every inventory beacon as part of the FlexNet Beacon code installation. However, there are preliminary configuration steps needed to allow remote execution to proceed.

To configure remote execution:

1. Ensure that you have the appropriate inventory beacons fully operational.

   To do this, navigate to **Discovery & Inventory > Beacons** (in the **Network** group), and check the following properties for an existing inventory beacon:

Property	Expected Value
Beacon status	`Operating normally`
Policy status	`Up to date`
Connectivity status	`Connected`

   To deploy and configure a new inventory beacon, click **Deploy a beacon**. Consult the online help for these pages for more information.

2. Ensure that at least one inventory beacon is configured to cover the subnet containing the target inventory devices.

   While all inventory beacons receive all rules declared in the web interface of FlexNet Manager Suite (when they download the `BeaconPolicy.xml` file), each one enacts only those rules that apply to target devices that fall within their assigned subnet(s). This setting is available through the web interface for FlexNet Manager Suite at **Discovery & Inventory > Beacons**. See the online help there for more information.

   Tip: *It is best practice to deploy an inventory beacon into each subnet that contains target inventory devices. This allows the inventory beacon to reliably use ARP or* `nbtstat` *requests to determine the MAC address of a discovered device (reliability of these results is reduced across separate subnets). Where, across subnets, only an IP address can be found for a device (that is, the device data is missing both a MAC address and a device name), a record is created for the discovered device; but because IP addresses may be dynamic, this is insufficient to allow merging with more complete records (which also contain either or both of the MAC address and a device name). Such complete discovery records may be created automatically when inventory is first returned from the locally-installed FlexNet inventory agent: not finding an existing, complete and matching discovered device record to link with the inventory device record, FlexNet Manager Suite automatically creates one. This means you may see multiple discovered device records with duplicate IP addresses: one record is complete (from inventory), and one or more others are missing identifying data (across subnets) as discussed. These cannot be merged automatically, and you are left with a manual task to clean up incomplete duplicate discovered device records. What's worse, if you have a rule to repeat the discovery process (for example, looking for newly-installed devices) and you still have incomplete discovery data from an inventory beacon reaching across subnet boundaries, the unmatched and incomplete record is recreated at each execution of the discovery rule.*
   In contrast, having a local inventory beacon in the same subnet as target devices provides both the IP address and the MAC address, which is sufficient for matching discovered device records. If you must do discovery across subnet boundaries without a local inventory beacon, ensure that there are full DNS entries visible to the inventory beacon for all devices you intend to discover. This allows the inventory beacon to report both an IP address and a name (either the device name or a fully-qualified domain name [FQDN]), which combination is again sufficient for record matching.

3. If yours is a highly secure, locked down environment, you may need to open network ports on the target computer devices to allow for remote execution.

Since the inventory beacons use standard ports to access target devices and remotely gather inventory, the required ports are already available in many environments. (The ports are documented in the online help, under *FlexNet Manager Suite Help* > *Inventory Beacons* > *Inventory Beacon Reference* > *Ports and URLs for Inventory Beacons*. The default requirements for remote execution are ports 445 for SMB on Windows and 22 for SSH on Unix.)

4. Ensure adequate credentials are available for the remote execution process to run. There are two possible approaches for Windows devices:

   - You can register a domain administrator account that has installation privileges on all the target computer devices within the domain. This approach minimizes entries in the Password Manager.
   - You can record appropriate (potentially unique) credentials for each device in the Password Manager. With this approach, you should also add filters to limit the number of password attempts on each target device, so that the remote execution attempt is not terminated because it attempted too many credentials without success.

   These credentials must be recorded in the secure Password Manager available on each inventory beacon (for details, see the online help, under *FlexNet Manager Suite Help* > *Inventory Beacons* > *Password Management Page*).

   For UNIX-like devices, the `ssh` daemon must be installed, and you must either:

   - Record `root` credentials for the target device in the Password Manager on the applicable inventory beacon
   - Record *non*-root credentials for the target device in the Password Manager on the applicable inventory beacon, and additionally ensure that a tool to allow privilege escalation (such as `sudo` or `priv`) is installed on target devices and either:
     a. The use of that tool is configured in the Password Manager (in the extra fields exposed when you specify and SSH account type), or
     b. Target devices are configured to allow escalation of privileges without requiring an interactive password.

5. Navigate to **Discovery & Inventory > Discovery and Inventory Rules**, and create one or more rules to take inventory from target computing devices within your enterprise, and then to collect inventory from them.

   Rules consist of:

   - **Targets** that identify sets of devices, and (for all the devices identified within a single target) specify policy about how to connect, whether to collect CAL evidence, whether to track application usage, and whether to adopt — for the Zero-footprint case, it is *critical* that either
     - The target device is *not* included in any target that has **Allow these targets to be adopted** selected; *or*
     - The target device *is* included in an active target for which **Do not allow these targets to be adopted** is selected (as a 'deny' always over-rides an 'allow'). This may be the easier condition to set and maintain over time.
   - **Actions** that declare what to do to the targeted devices — to ensure discovery and inventory collection, the relevant action must ensure that, in the **General devices discovery and inventory** section (click the title bar to expand the section), one or both of the **Discover ...** check boxes is selected, and **Gather hardware and software inventory** is selected. In addition, other specialized kinds of inventory may be selected, depending on the target inventory device. By specifying multiple rules, you can customize actions to gather all required inventory types while minimizing activity on any individual target device.

- A schedule for implementing the action on the targeted devices.

For more information, see the online help on these product pages.

When these preliminary configuration steps are in place, the inventory beacon collects inventory from target devices in its assigned subnet on the schedule declared in the relevant rule(s). The details of the execution process are included in Zero-Footprint: Normal Operation.

Zero-Footprint: Troubleshooting Inventory

Inventory gathering and upload is a sophisticated chain from target inventory device through inventory beacon to central application server. For general trouble-shooting over the whole process, see the online help for FlexNet Manager Suite under *Inventory Beacons* > *Inventory Beacon Reference* > *Troubleshooting: Inventory Not Uploading*. This topic focuses entirely on inventory collection on the target inventory device.

When you use Zero-footprint inventory collection, the normal log files for the `ndtrack` executable are saved on the target inventory device:

- On Windows platforms, in `C:\Windows\temp\ManageSoft\tracker.log`
- On UNIX-like platforms, in `/var/tmp/flexera/log` (when the executable was invoked by the `root` account, as recommended) or `/var/tmp/flexera.`*`UserName`*`/log` (when invoked by the *`UserName`* account).

For advanced trouble-shooting, you may require more advanced tracing and logging. You may also be asked to submit a trace file to assist the Support team at Flexera to solve difficult problems in your environment.

By default, tracing is not available with Zero-footprint inventory gathering. However, with some custom preparation, you can set up for, and control, tracing with a `.trace` configuration file of your own. Even though Zero-footprint inventory gathering is triggered from the inventory beacon, the configuration file and the resulting log(s) all reside on the target inventory device.

To set up and configure tracing for Zero-footprint inventory gathering:

1. Obtain a copy of an `etcp.trace` file from an installed FlexNet inventory agent on another device.

   Where the full FlexNet inventory agent has been installed on a target device, the `etcp.trace` file is located with the `ndtrack` executable:

   - On Windows, the default is `C:\Program Files (x86)\ManageSoft\etcp.trace`
   - On UNIX-like platforms, the default is `/opt/managesoft/etcp.trace`.

   Because this file format is consistent across platforms, you may take your copy of `etcp.trace` from any inventory device where the full FlexNet inventory agent is installed.

2. On a Windows target device:

   a. Save the `etcp.trace` file in `C:\Temp\`.

3. Configure the name and location of the trace/log file that will be generated on the inventory device.

   The hash or pound character (#) identifies a comment. To "uncomment" a line in the `.trace` configuration file means to delete (only) the leading hash character. Choose one of the following lines, uncomment it, and

optionally modify it to your requirements. On Windows:

```
#filename=C:\ManageSoft.log
#filename=C:\ManageSoft%p_%d_%t_%u.log        # filename pattern with everything!
```

On UNIX-like platforms:

```
#filename=/tmp/log/mgstrace.log
#filename=/tmp/log/ManageSoft%p_%d_%t_%u.log # filename pattern with everything!
```

See the notes within the file header for the use of the supported variables within the file name.

Tip: *It is best practice to use a pattern that includes (at least) either a date stamp (%d) or a sequential number (%u). Without these, the fixed file name means tracing information is appended to the same file with every inventory collection. This can quickly produce a trace file too large for text editors to read, and too hard to manage in terms of disk space. Variables in the file name trigger creation of a new file each time the associated variable is changed (or, for %u, at every invocation of `ndtrack`).*

Important: *The log file path:*

- *Must be on the same drive as the `ndtrack` executable (on Windows devices)*
- *Must exist and be writable before the `ndtrack` executable is next invoked (tracing does not create any directories, and does not function if any directory in the specified path is missing or unwritable).*

4. Uncomment the lines for which you want to enable tracing (ensuring that the uncommented line now starts with a plus sign).

   For Zero-footprint inventory gathering, the typical lines to uncomment are:

   ```
   +Inventory
   +Error
   +Communication/Network
   ```

When Zero-footprint inventory gathering is invoked on the inventory beacon, it creates the log file on the target inventory device as you specified, ready for your inspection.

5

FlexNet Inventory Scanner: Details

This chapter provides great detail about the FlexNet inventory core components when these are deployed as self-extracting executables (also known as the FlexNet Inventory Scanner). As part of the functionality available in this case, the operational executables are removed after each execution, although the FlexNet Inventory Scanner package itself is not automatically removed.

In this configuration, and given appropriate command lines, the FlexNet inventory core components are capable of collecting hardware and software inventory from a target device, and uploading it to a location specified in the command line. This configuration is often useful for pilot studies and test cases. For details of the distinct use cases, refer back to Understanding What, Where, How, and Why.

This document provides a consistent set of data (as far as possible) across all the different use cases, each in its own chapter. This means that, once you have chosen your preferred use case, you can focus only on the details for that one, and ignore all other use case chapters.

In addition to the distinct chapters for the different use cases, you should also review the subsequent chapter on functionality that is common throughout. This is followed by detailed reference material on command lines, preferences, file formats, and the like.

FlexNet Inventory Scanner: Normal Operation

The regular operation of the FlexNet inventory core components in the FlexNet Inventory Scanner configuration have a number of differences across Windows and non-Windows platforms. For simpler reading, each is treated separately in one of the following topics.

For details about deployment and implementation, see FlexNet Inventory Scanner: Implementation and appropriate subtopics.

FlexNet Inventory Scanner: Operation on Windows

This description assumes that you have obtained and appropriately deployed the FlexNet Inventory Scanner to working locations (see FlexNet Inventory Scanner: Implementation on Windows).

The FlexNet Inventory Scanner may be invoked through a command line (details are available in ndtrack Command Line), through a scheduled task (although this is not generally recommended), or simply by double-clicking the executable file while you are logged on using an account with suitable administrator privileges.

At each invocation, the self-installing executable:

1. Installs the Windows version of the tracker (or inventory collection) executable (`ndtrack.exe`) and related control files in the `temp` directory of the account that is executing it.
2. Executes `ndtrack`, either according to the command-line options supplied, or using its default parameters. This means that the tracker component runs as the same account that executed the FlexNet Inventory Scanner.

   Running as the `LocalSystem` account is recommended, because elevated privileges are required to complete several aspects of inventory gathering. It is *possible* to run the tracker under a non-`LocalSystem` account, but best practice is to run it with administrator privileges, or you may lose inventory functionality.

   Note: *On Microsoft Windows, the tracker does not prevent invocation by an account that has lesser privileges; but you would then need to ensure that such an account had all the required access rights for the kinds of inventory you expected to gather on a target device. Since this is highly dependent on your environment, this approach is unsupported.*

   When no command-line options are supplied, the default behaviors are:

   - The tracker gathers a machine inventory on the local computer on which it is currently running.
   - It saves this inventory in `%temp%\FlexeraSoftware\$(UserName) on $(MachineId).ndi`, where `%temp%` is the temporary directory for the account that is running the FlexNet Inventory Scanner, with the file name showing the account and machine ID related to the inventory run.
   - Returns an exit code of `0` for success. (For any other exit code, check the log file.)
   - Records the log for this activity in `%temp%\ManageSoft\tracker.log`.
   - If `InventorySettings.xml` is supplied and `ndtrack` can connect to the Oracle Database, returns a separate `.ndi` Oracle inventory file. (Details about `InventorySettings.xml` are included in FlexNet Inventory Scanner: Implementation on Windows. For details of account setup for gathering Oracle inventory, see *Credentials for FlexNet Inventory Scanner Inventory* in *FlexNet Manager Suite System Reference*.)
   - If the Oracle Net Listener can be found and queried, returns a discovery file with details of the Oracle listener.

   These behaviors can be modified with command-line options for `ndtrack`, attached to the command line for FlexNet Inventory Scanner (which passes them directly through to `ndtrack`). For example, you can configure the FlexNet Inventory Scanner to trigger an immediate upload to an inventory beacon of your choice for integration into the standard inventory processes. For details of these command-line options, see ndtrack Command Line.
3. When execution is completed, FlexNet Inventory Scanner uninstalls the `ndtrack` executable, leaving only the self-installing executable `FlexNet Inventory Scanner.exe` in place.

FlexNet Inventory Scanner: Operation on UNIX-Like Platforms

This description assumes that you have deployed `ndtrack.sh`, optionally its customized configuration file `ndtrack.ini`, and the current and unedited `InventorySettings.xml` (required when Oracle inventory is in play). These processes are described in FlexNet Inventory Scanner: Implementation on Unix-Like Platforms.

For normal operation, do either of the following:

- To run `ndtrack.sh` from the command line, it's convenient to change directory to the location of these files (all in the same folder), and invoke the executable together with any further command-line parameters (documented in FlexNet Inventory Scanner Command Line).
- To run the lightweight scanner regularly, configure your preferred scheduling tool (such as `cron`) on the target device to regularly invoke `ndtrack.sh`, either using the options available in FlexNet Inventory Scanner Command Line or using a customized copy of `ndtrack.ini` (see Configuring ndtrack.ini for UNIX-like Platforms). To execute the scanner, you can either use the `chmod` command to set the execute permissions on `ndtrack.sh`; or you can start `ndtrack.sh` using a shell (such as `/bin/sh /path/ndtrack.sh [options]`.

Tip: *Experienced administrators can also combine use of `ndtrack.ini` (for common defaults) with matching command-line parameters (for local overrides on specific devices).*

You may execute `ndtrack.sh` either as `root` (recommended), or as a different user. Running as a non-`root` user limits the effectiveness of inventory gathering, as described in FlexNet Inventory Scanner: Accounts and Privileges.

At each invocation, `ndtrack.sh`:

1. First determines the current operating system, and extracts the binaries appropriate to that platform to a unique folder under the home directory of the current user, provided that this has sufficient space and is not the root directory (`/`). If that is unsuccessful, it uses the `/var/tmp` folder.

   If there is insufficient space in `/var/tmp`, `ndtrack.sh` writes an error to `stderr`, and the process stops.

2. Checks in its installed folder for a customized `ndtrack.ini` configuration file.

   If found, this file in its entirety replaces all the default values for operating preferences and data providers (see Configuring ndtrack.ini for UNIX-like Platforms). Take care that the configuration file is complete, in case missing parameters are effectively 'turned off'.

3. Uses any command-line parameters to override matching values (whether built-in defaults, or values from `ndtrack.ini`).

4. Executes the inventory collection according to the resulting set of preferences.

When neither command-line options nor preferences in `ndtrack.ini` are specified, `ndtrack.sh` does the following:

- Conducts a machine inventory on the local computer on which it is currently running. (Only machine inventory is supported for UNIX-like platforms: there has never been any equivalent of the "user"-based inventory available on Microsoft Windows for backward compatibility.)
- Creates an inventory file named `$(UserName) on $(MachineId).ndi`, where these variables are replaced as follows:

  - *$(UserName)* is replaced by the name of the account running the executable. When ndtrack is run as root, this value is returned as system, for consistency with inventory reported from Windows platforms.
  - *$(MachineId)* is replaced by the computer name of the inventory device, as returned from the operating system.
- Saves this inventory file (both uncompressed for inspection and compressed for upload) in either of the following paths:
  - When the executable has run as the root account, in /var/tmp/flexera/uploads/inventories
  - When the executable has been run by any other user account (represented as *UserName*), in /var/tmp/flexera.*UserName*/uploads/inventories.
- Returns an exit code of 0 for success. For any other exit code, check the log file.
- Where InventorySettings.xml is supplied and ndtrack is running as root and can connect to an instance of Oracle Database, creates a separate Oracle inventory file (such as *$(MachineId)* at *DateTimeInISO8601* (Oracle).ndi.gz) in the same inventories folder.
- Where the Oracle listener is found and can be queried (by ndtrack running as root), creates a .disco discovery file containing details of the local Oracle Net Listener (such as *$(MachineId)* at *DateTimeInISO8601*.disco), saved in /var/tmp/flexera/uploads/Discovery.
- Records the log for these activities in tracker.log in either of the following locations:
  - When the executable has run as the root account, in /var/tmp/flexera/log
  - When the executable has been run by any other user account (represented as *UserName*), in /var/tmp/flexera.*UserName*/log.

Tip: *Discovery of Oracle Net Listener and collection of inventory from Oracle database instances are both blocked unless the tracker component (ndtrack executable) invoked by the FlexNet Inventory Scanner is running as root.*

These behaviors can be changed with command-line parameters described in FlexNet Inventory Scanner Command Line, or preferences saved in ndtrack.ini (see Configuring ndtrack.ini for UNIX-like Platforms). As a customization example, you can configure ndtrack.sh to upload the resulting inventory file immediately to an inventory beacon of your choice for integration into the standard inventory processes.

FlexNet Inventory Scanner: System Requirements

The following details apply to the FlexNet inventory core components when deployed as a self-extracting executable to a target device (or where applicable to a file share).

Supported platforms

The FlexNet inventory core components operate on the following platforms (inventory targets):

Microsoft Windows	UNIX-like platforms

• Windows Server 2008 R2 x64, 2012, 2012 R2, 2012 R2 SP1, 2016, 2019, 2022
• Windows Server Core 2008, 2008 R2 x64, 2012, 2012 R2
• Windows Server Standard (previously known as Windows Server Core) 2016, 2019
• Windows Vista, 7, 8, 10, 11
 	
Note: *Windows systems that can run on ARM-based devices are also supported.*
	• AIX 7.1 LPARs with Technology Level 5 or later, AIX 7.2
	• Amazon Linux 2, 2023 (ARM64/AArch64; x86, 32-bit and 64-bit)
	• CentOS 6-7.5 (x86, 32-bit and 64-bit); 7.6-9 (x86 64-bit only)
	• Debian Linux 8–11.3 (x86, 32-bit and 64-bit); 11.5, 12.0 (x86 64-bit only)
	
	Note: *For Debian Linux 9.4, 9.5 and 10 (both 32-bit and 64-bit), minimal installations of the OS core omit the `ifconfig` command. This prevents collection of the IP address and MAC address in inventory. The `root` user can use the following command to restore this functionality:*
	`apt-get install net-tools -y`
	• Fedora Linux 26 (x86, 32-bit and 64-bit); 27-38 (x86 64-bit only)
	• HP-UX 11i v3 (Itanium only), vPars/nPars
	• macOS 10.15.4–14 (applicable for both Intel and Apple M-series processors)
	
	Note: *If you need to run the FlexNet inventory agent of version 20.1 or earlier on an Apple M-series processor ("Apple silicon"), Rosetta 2 must be installed and running. This is Apple's solution for transitioning most Intel-based applications to run on Apple silicon. There are two possible command formats for installing Rosetta 2:*
	◦ *Interactive installation that asks for agreement to the Rosetta 2 license:*
	`/usr/sbin/softwareupdate --install-rosetta`
	◦ *Non-interactive installation:*
	`/usr/sbin/softwareupdate --install-rosetta --agree-to-license`

Microsoft Windows	UNIX-like platforms
	• Nutanix AHV hypervisor ***Note:*** *All versions that are within their product support lifecycle and use the* `Libvirt` *library are supported.* • OpenSUSE Leap 42.2, 42.3 (x86, 32-bit and 64-bit); 15-15.4 (x86 64-bit only) • Oracle Linux 6.0–6.10 (x86, 32-bit and 64-bit); 7.0-8.7 and 9.0-9.1 (x86 64-bit only) • Photon OS 3.0-5.0 • Red Hat Enterprise Linux (RHEL) 6.0-6.10 (x86, 32-bit and 64-bit); 7.0-8.8 and 9.0-9.2 (x86 64-bit only) • Rocky Linux 8, 9 • Solaris 10–11 (SPARC), Zones for versions 10–11 • Solaris 10–11.4 (x86), Zones for versions 10–11 • SUSE Linux Enterprise Server 12 SP3, 12 SP4, 12 SP5, 15, 15 SP1, 15.2-15.5 (x86 64-bit only) • Ubuntu 14–17.04 (x86, 32-bit and 64-bit); 17.10-23.04 (x86 64-bit only) Linux on IBM zSystems platforms are also supported. The following Linux distributions are certified for usage on the IBM zSystems architecture. For details about supported versions on different hardware types, see https://www.ibm.com/support/pages/linux-ibm-z-tested-platforms. • Red Hat Enterprise Linux • SUSE Linux Enterprise Server • Ubuntu

Disk space requirements

On target inventory devices in the FlexNet Inventory Scanner case, the following are platform-specific disk space requirements:

- Windows: Where the target device is a typical workstation, in the order of 2MB; and for an Oracle server, in the order of 5MB.
- UNIX-like platforms: The downloaded code is approaching 23MB, and it then looks for an additional 16MB of working

disk space in either (in priority order):

1. The home directory of the account running the inventory (unless that directory is `/`)
2. `/var/tmp`.

Where this space is not available, inventory collection is not attempted.

After inventory collection in the FlexNet Inventory Scanner case, the following (non-executable) files are left on the target inventory device as a potential aid to process validation or trouble-shooting, and are all replaced at the next iteration of the same process:

- An uncompressed inventory (`.ndi`) file is left:
  - For Windows devices, in `%TEMP%\FlexeraSoftware` (that is, in a subdirectory of the temporary directory for the account running the FlexNet Inventory Scanner)
  - For UNIX-like devices, when inventory collection is (as usual) run as `root`, in `/var/tmp/flexera/tracker`; or if inventory collection is run as another user, in `/var/tmp/flexera.`*`userName`*`/tracker`.
- Log files are saved on the target inventory device:
  - For Windows devices, in `C:\Windows\temp\ManageSoft`
  - For UNIX-like devices, when inventory collection is (as usual) run as `root`, in `/var/tmp/flexera/log`; or if inventory collection is run as another user, in `/var/tmp/flexera.`*`userName`*`/log`.

The following log file is available on the target inventory device in the FlexNet Inventory Scanner case:

- `tracker.log` — Generated by the inventory component, `ndtrack`

Memory requirements

On target inventory devices, the memory requirements are:

- Minimum RAM: 512 MB
- Recommended RAM: 2 GB

In general, through a cycle of inventory gathering and upload, the memory demand is in the order of 5-30 MB.

Communications protocols and ports

In the network layer, both IPv4 and IPv6 formats are supported for communications between the FlexNet Inventory Scanner and an inventory beacon. This applies across both Windows (with `FlexeraInventoryScanner.exe`) and UNIX-like platforms (with `ndtrack.sh`). For details on the architectural impacts, see Support for IPv6 Networks.

When the FlexNet inventory core components execute on the target device (in the FlexNet Inventory Scanner case), the only ports required are the standard ports for communication with the inventory beacon:

- Windows and UNIX: From FlexNet inventory core components on target device inbound on the inventory beacon — File upload using HTTP protocol: port 80
- Windows only: From FlexNet inventory core components on target device inbound on the inventory beacon — File upload using HTTPS protocol: port 443

Tip: *HTTPS is not supported for UNIX-like platforms in the FlexNet Inventory Scanner case.*

Supported packages to inventory

FlexNet inventory can include data from most package technologies supported by the operating systems, and some additional third-party packaging technologies:

Platform	Supported package technologies
All platforms	InstallAnywhere (IA), InstallShield Multiplatform (ISMP), BEA/Oracle Installer (BEA), Oracle Universal Installer (OUI), IBM Installation Manager (IIM)
AIX	LPP, RPM
HP-UX	Software Distributor SD-UX Package
Linux	RPM (Red Hat, CentOS, Oracle, SuSE, Fedora, etc), DPKG (Debian, Ubuntu).
Mac OS X	Mac Application Bundle, Mac Package Bundle
Solaris	Sys V Package (pkg), IPS
Windows	MSI, Add/Remove Programs Registry Key

However, FlexNet inventory cannot collect data from some of the less common or newer operating system technologies and many third-party technologies. Some known examples include:

- All platforms — IBM InstallStream, IBM Tivoli Netcool Installer
- Mac OS X — Mac flat package.

System load benchmarks

The following notes reflect observed behavior on sample target inventory devices during collection of FlexNet inventory:

Task	Run duration (seconds)	CPU usage (seconds)	CPU usage (% of single core)	Memory usage	Network load
Inventory collection	13 to 240 s	5 to 130 s	10% to 50%	4 MB to 20 MB	10 KB to 200 KB per upload

FlexNet Inventory Scanner: Accounts and Privileges

In the FlexNet Inventory Scanner case, when the FlexNet inventory core components (run locally as the FlexNet Inventory Scanner) gather hardware and software inventory from the target inventory device, they execute as the user name (or account) that triggered the execution.

To effectively gather inventory, this account must have elevated privileges:

- On Windows, it must have administrator privileges on the local machine.
- To collect complete inventory on UNIX-like platforms, you must elevate privileges (for example, with `sudo` or `priv`) and execute `ndtrack.sh` as `root`.

If `ndtrack.sh` executes as a non-root user on UNIX-like systems, the following are amongst the inventory details that *cannot* be collected:

- File evidence from any file system path not accessible by the executing user
- InstallAnywhere, InstallShield Multiplatform, or Oracle Universal Installer evidence under paths not accessible by the executing user
- Oracle Database service discovery via the local listener using `lsnrctl`
- Oracle Database inventory which may use impersonation of an Oracle Database administration user when running the `sqlplus` command
- On Linux systems:
  - BIOS details (`dmidecode`): serial number, UUID, manufacturer, model, chassis type
  - All hard disk information (from device files)
- On Solaris systems:
  - MAC addresses of network adapters
  - x86 BIOS details (`dmidecode`): model, manufacturer
  - SPARC model using OpenPROM interface (the tracker fails over to using the value from `sysinfo SI_PLATFORM` instead, which may give different results)
- On HP-UX systems:
  - SD-UX installation evidence from `swlist` if access has been locked down with `swreg` or `swacl`
  - vPar evidence including `VMType`, `VMName`, and vPar capacity (`vparstatus` requires root)
  - Hard disk drive properties including capacity
- On Mac OS X systems:
  - Mac OS X package bundle paths under `/Applications` or `/System/Library` not accessible by the executing user.

FlexNet Inventory Scanner: Implementation

The processes for obtaining, configuring, deploying, and using the FlexNet inventory core components in the FlexNet Inventory Scanner configuration have a number of differences across Windows and non-Windows platforms. For simpler reading, each is treated separately in one of the following topics.

Those topics are followed by details of configuration.

There are two ways to configure the inventory agent on inventory devices:

- At run-time, using command-line options (see FlexNet Inventory Scanner Command Line).
- Use the configuration file specific to the platform:
  - On Microsoft Windows, use the `wmitrack.ini` file that governs the Windows Management Instrumentation class tracking behavior. This is described in Configuring WMI for FlexNet Inventory Scanner.
  - On UNIX-like platforms, use the `ndtrack.ini` file that (like the `wmitrack.ini` file on Windows) controls the providers used for various kinds of inventory gathering; but unlike the other, can also be extended to include any of the preferences that may be set on the command-line (see Configuring ndtrack.ini for UNIX-like Platforms).

Note: *On Microsoft Windows, the FlexNet Inventory Scanner is different than the installed FlexNet inventory agent (where an inventory target has been 'adopted' by the local installation of the full inventory agent) in this regard:*

- *The lightweight FlexNet Inventory Scanner does not access Windows registry settings to determine options.*
- *The full FlexNet inventory agent may access a wide range of registry settings to control its behavior.*

FlexNet Inventory Scanner: Implementation on Windows

This process covers obtaining, configuring, and deploying the FlexNet inventory core components in the FlexNet Inventory Scanner configuration on Microsoft Windows platforms. Two forms of deployment are covered:

- A one-off manual deployment for testing, evaluation, or simple infrastructures
- A repeatable process for deploying the FlexNet Inventory Scanner to multiple devices.

Tip: *It is best practice for a user account with local administrator privileges to execute the FlexNet Inventory Scanner. While the agent still runs with lesser privileges, it only collects the full range of inventory when executed with administrator privileges. Therefore, if you are about to do a one-off, manual deployment, it is helpful to log into the target device using a local administrator account.*

You can conveniently begin this process on a server where you want to deploy the FlexNet Inventory Scanner.

To install and run the FlexNet Inventory Scanner:

1. Use your browser to access the Flexera Customer Community.
   a. On https://community.flexera.com/, use the account details emailed to you with your order confirmation from Flexera to log in (using the **Login** link in the top right).

      Tip: *Access requires your Customer Community user name and password. If you do not have one, click the `Let's go!` button on the login page to request one. Your credentials are configured for access to content you have licensed.*

   b. Select **Find My Product** and choose **FlexNet Manager** from the top menu. Now click the button

PRODUCT RESOURCES - PRODUCT INFORMATION which will expose the **Download Products and Licenses** link. Click on this option.

A routing page appears to let you `Access Product and License Center`, displaying lists of products from Flexera.

c. In the lists of products, identify `FlexNet Manager Platform`, and immediately below it, click **LET'S GO**.

The `Product and License Center` site displays.

d. In the `Your Downloads` section of the `Home` page, click the link for FlexNet Manager Platform.

e. In the `Download Packages` page, click the link for FlexNet Manager Platform 2023 R2 to access the downloads.

2. Select the **Flexera Inventory Scanner for FlexNet Manager Suite 2023 R2** archive (`Flexera Inventory Scanner for FlexNet Manager Suite 2023 R2.zip`), and download to a convenient location (such as `C:\temp`).

3. Expand (unzip) the archive and save `FlexeraInventoryScanner.exe` to your preferred path.

A typical path is in the `temp` folder for the logged-in account (or, if you prefer, a subfolder such as `temp\FIS`). When the self-installing executable is run, it installs the operational code into the current user's `temp` folder, making it convenient if both the self-installing executable and the operational code objects are in the same folder (or one closely related).

***Tip:** Optionally, you may save `FlexeraInventoryScanner.exe` to a file share that is accessible from many target devices.*

4. Optionally, customize the WMI classes used for inventory collection on this device.

The inventory tool within the FlexNet Inventory Scanner provides defaults for normal operation. You may override and extend these default behaviors by providing a customized `wmitrack.ini` in the same folder where the self-installing executable is saved (suggested: the `temp\FIS` folder). Notice that this is not the operational folder where the tools execute, but the storage location of `FlexeraInventoryScanner.exe`. If a customized `wmitrack.ini` file is present, it replaces *all* the default values used by the inventory tool when it is installed, so be careful not to omit any settings that you may require from your customized file. To customize:

a. Log into an inventory beacon, navigate to `%ProgramFiles%\Flexera Software\Inventory Beacon\Tracker`, and take a copy of `wmitrack.ini` to the device where you are working.

b. Edit your new copy of `wmitrack.ini` to suit your requirements.

For more information, see WMI Configuration File (wmitrack.ini).

c. Save the modified file with the same `wmitrack.ini` file name in the same folder where you stored `FlexeraInventoryScanner.exe` (suggested: the `temp\FIS` folder).

5. Optionally, extend the inventory-gathering of the FlexNet Inventory Scanner with additional functionality you have licensed.

Functionality extensions are embedded in the file `InventorySettings.xml`, which may be updated from time to time in the ARL download process. Your current license terms are integrated with the same file on the

central application server, and the result is automatically downloaded to your inventory beacons after each update. On the inventory beacon, the file is saved in `%CommonAppData%\Flexera Software\Beacon\InventorySettings`. Particularly if

a. You have licensed the FlexNet Manager for Datacenters product, and

b. You want this instance of FlexNet Inventory Scanner to collect Oracle inventory, or Microsoft SQL Server inventory, from the target device,

copy `InventorySettings.xml` from an inventory beacon to the target device, into the same folder where you stored `FlexeraInventoryScanner.exe` (suggested: the `temp\FIS` folder).

Important: *Do not edit this file!*

Tip: *Take note of the revision number in the first line of the `InventorySettings.xml` file for future reference. Set a reminder for yourself in your preferred tool to check the revision number of `InventorySettings.xml` after future updates of the Application Recognition Library. If the revision changes, you should manually update all the copies you manually deployed. These changes allow the inventory tool to stay abreast of inventory changes necessary to support the latest Oracle or SQL Server licenses.*

6. To run a test and inspect the resulting inventory files, double-click the self-installing `FlexeraInventoryScanner.exe` in Windows Explorer.

Remember: *For complete hardware and software inventory collection, you should be logged in with local administrator privileges.*

The installer copies the executables, together with any co-located configuration files (`wmitrack.ini` and `InventorySettings.xml`), into the `temp` directory of the current user account on the target device, and immediately initiates inventory collection. When started in this way, there are no settings for uploading the result, and the uncompressed inventory (`.ndi`) file(s) are left in the user's `temp` directory. You may open an `.ndi` file in a text editor to review its contents.

7. To run once and upload the resulting inventory files:

   a. Using your account with administrator privileges, open a command window on the device.

   b. Navigate to the folder containing the self-installing executable (suggested: the `temp\FIS` folder).

   c. Execute the FlexNet Inventory Scanner with command-line options for inventory upload, such as the following (all on one line):

```
FlexeraInventoryScanner.exe
    -o UploadLocation="http://InventoryBeacon/ManageSoftRL"
    -o Upload="true"
```

   substituting the fully qualified domain name of your inventory beacon for the placeholder *`InventoryBeacon`*. (`ManageSoftRL` is the name of a web service on the inventory beacon that receives the uploaded inventory and saves it by default to *`%CommonAppData%`*`\Flexera Software\Incoming\Inventories`.) Keep in mind that the `-o Upload="true"` option (including enclosing the value in double quotation marks) is mandatory for uploads by this executable on the Windows platform.

Any command-line options you provide to the self-installing executable are passed directly through to the inventory tool (`ndtrack.exe`). For details of other available options, see ndtrack Command Line. Immediately after completing the inventory collection, the inventory tool attempts to compress and upload the inventory file(s) to the inventory beacon you identified.

Tip: *Because the FlexNet Inventory Scanner does not include the separate `ndupLoad` component, there is no retry in the event of the failure of this one-time upload attempt.*

When the initial upload attempt is successful, the inventory file(s) are collected with all others uploaded to this inventory beacon, and join the standard process of upload to, and resolution on, the central application server. When the process is complete, the executables (and any optional configuration files) are uninstalled from the user's `temp` directory. As always, uncompressed copies of the `.ndi` files are saved in the `temp` directory of the executing account on the target device, and replaced at each future repeat of this process.

Note: *The purpose of the FlexNet Inventory Scanner is as described above: one-time use in special circumstances, including for testing during infrastructure development. It is not intended for full-time production use, as it lacks facilities like upload retries, policy updates, self-updates, and the like that are needed in a production environment. For production use, best practice is to deploy the full FlexNet inventory agent (in either the Adopted case or the Agent third-party deployment case). At the very least, it is preferable to deploy the FlexNet inventory core components (in the Core deployment case) to avoid the repeated overheads of installation and removal of the executable code by the self-installing executable `FlexeraInventoryScanner.exe`. The following steps, then, are not recommended, but provided only as thought starters if you intend to deploy FlexNet Inventory Scanner.*

To deploy the FlexNet Inventory Scanner more widely:

8. If you choose to deploy the FlexNet Inventory Scanner more widely:

   a. Package the `FlexeraInventoryScanner.exe`, along with the optional `wmitrack.ini` and `InventorySettings.xml` files (when applicable), in your preferred deployment tool.

   b. Deploy these to known locations on target devices.

      One location to consider is on a file share accessible from many target devices.

   c. As part of the deployment process, set up a scheduled task in your preferred scheduling tool (such as Microsoft Scheduler) on each target device:

      - The task should run under an account with administrator privileges, preferably the local `SYSTEM` account.
      - The command line must include parameters to attempt uploads to an inventory beacon (preferably a high-reliability server and network, since there is no catch-up from failed upload attempts), such as the following (on a single line, and inserting your inventory beacon server's domain name in place of the place-holder):

        ```
        drive-and-path\FlexeraInventoryScanner.exe
            -o UploadLocation="http://InventoryBeacon/ManageSoftRL"
            -o Upload="true"
        ```

- The command line may include other parameters discussed in ndtrack Command Line.
- You may need to vary (or randomize) the scheduled time for inventory collection, to spread the load on either the file share hosting the self-installing executable or the inventory beacon accepting the subsequent uploads.

Configuring WMI for FlexNet Inventory Scanner

The behavior of FlexNet Inventory Scanner on Windows devices can (in part) be configured in a `wmitrack.ini` file.

The WMI (Windows Management Instrumentation) configuration file is used to inform the inventory agent what hardware, software and operating system components it should track. This file is only used if WMI tracking is selected as the preferred mechanism for hardware inventory tracking (set by the `WMI` preference, for which see FlexNet Inventory Scanner Command Line).

The components to be tracked can be any valid Win32 classes. A full list of supported classes is provided through the following URLs:

- Win32 classes: http://msdn.microsoft.com/en-us/library/aa394084(v=vs.85).aspx
- CIM classes: http://msdn.microsoft.com/en-us/library/aa386179(v=vs.85).aspx
- Software licensing classes: http://msdn.microsoft.com/en-us/library/ee957720(v=vs.85).aspx

You can edit this text-based file to change the items being tracked. For example, if you want to track user logons, edit the `wmitrack.ini` file to include the lines:

```
[Win32_ComputerSystem]
     UserName
```

The `UserName` value returns the name of the user currently logged on. In a terminal services session, `UserName` returns the name of the user that is logged on to the console — not the user logged on during the terminal service session. `UserName` may be null if the user currently logged on does not have administrative privileges. If you experience this problem, try using;

```
[Win32_LoggedOnUser]
     Antecedent
```

The user data collected will be available in hardware inventory reports.

You can also exclude an entire section, or an individual item, by commenting them out with a leading semi-colon. For example, in the sample file, see

```
;[Win32_USBDevice]
```

Locations

For remote inventory collection from Windows devices, the FlexNet Inventory Scanner checks the following in this order:

1. It looks for a command-line option `-o WMIConfigFile= "`*`FullPathAndFileName`*`"`, and uses the file declared there (and no further checking occurs).

Note: *There is no limitation on the file name or extension you may specify with this option.*

2. In the absence of a command-line option, FlexNet Inventory Scanner looks for a `wmitrack.ini` file in the same folder where the self-executing zip expanded (`%temp%\FlexeraInventoryScanner`), on the computer where the scanner is executing. This is the standard operating location for this `ini` file.

If a `wmitrack.ini` file is not found in either way, no WMI hardware components are tracked.

Sample file

The following default `wmitrack.ini` file specifies the Win32 items collected during standard inventory tracking. (You may use this as source for modifying your own alternative configuration file.)

```
[Win32_ComputerSystem]
Manufacturer
Model
Domain
DomainRole
NumberOfProcessors
NumberOfLogicalProcessors
TotalPhysicalMemory
Status
UserName

[Win32_ComputerSystemProduct]
IdentifyingNumber
Name
UUID
Vendor
Version

[Win32_OperatingSystem]
Name
Manufacturer
Version
ServicePackMajorVersion
ServicePackMinorVersion
SerialNumber
InstallDate
LastBootUpTime
OSLanguage
FreePhysicalMemory
FreeVirtualMemory
CountryCode
WindowsDirectory
SystemDirectory
Caption
CSDVersion
Status
```

```
CSName
OSType
OSArchitecture

[Win32_BIOS]
Manufacturer
Version
ReleaseDate
SerialNumber
BiosCharacteristics
Status

[Win32_Processor]
Description
Manufacturer
Version
ProcessorId
CurrentClockSpeed
CurrentVoltage
L2CacheSize
Status
MaxClockSpeed
Name
ProcessorType
NumberOfLogicalProcessors
NumberOfCores
DeviceID

[Win32_DiskDrive]
Description
Manufacturer
Model
Size
InterfaceType
Partitions
Status

[Win32_LogicalDisk]
Description
VolumeName
FileSystem
FreeSpace
Size
VolumeSerialNumber
DriveType
MediaType
Status
ProviderName
```

```
[Win32_CDROMDrive]
Description
Manufacturer
Drive
Status
Capabilities

[Win32_NetworkAdapter]
Manufacturer
MACAddress
MaxSpeed
Speed
Status

[Win32_NetworkAdapterConfiguration]
Caption
Description
Index
MACAddress
IPEnabled
DHCPEnabled
IPAddress
DHCPServer
DNSHostName
DNSDomain
DNSServerSearchOrder
DefaultIPGateway
IPSubnet

[Win32_PhysicalMemory]
Capacity
MemoryType
PositionInRow
Speed
Status

[Win32_SoundDevice]
Name
Manufacturer

[Win32_VideoController]
Name
VideoProcessor
DriverVersion
DriverDate
InstalledDisplayDrivers
AdapterRAM
```

```
[Win32_VideoConfiguration]
AdapterRAM
AdapterType
Description
HorizontalResolution
MonitorManufacturer
MonitorType
Name
VerticalResolution

[Win32_SystemEnclosure]

;[Win32_USBDevice]
;Caption
;ClassGuid
;Description
;DeviceID
;Manufacturer
;Name
;Status
;SystemName

[SoftwareLicensingProduct]
ApplicationID
Description
EvaluationEndDate
GracePeriodRemaining
LicenseStatus
MachineURL
Name
OfflineInstallationId
PartialProductKey
ProcessorURL
ProductKeyID
ProductKeyURL
UseLicenseURL

[SoftwareLicensingService]
ClientMachineID
IsKeyManagementServiceMachine
KeyManagementServiceCurrentCount
KeyManagementServiceMachine
KeyManagementServiceProductKeyID
PolicyCacheRefreshRequired
RequiredClientCount
Version
VLActivationInterval
```

```
VLRenewalInterval
```

FlexNet Inventory Scanner: Implementation on Unix-Like Platforms

This process covers obtaining, configuring, and deploying the FlexNet inventory core components in the FlexNet Inventory Scanner configuration on UNIX-like platforms.

For UNIX-like platforms, there is no conveniently pre-packaged, downloadable form of the FlexNet inventory core components specifically branded as the FlexNet Inventory Scanner. However, the following process allows you to obtain equivalent functionality.

Tip: *The required files are available only after the inventory beacon has downloaded its settings from the central application server.*

To obtain the FlexNet Inventory Scanner for UNIX-like platforms:

1. Log into a registered inventory beacon, and navigate to the default path `%ProgramFiles%\Flexera Software\Inventory Beacon\RemoteExecution\Public\Inventory` (defined in the Windows share `mgsRET$`) .
2. From this folder, collect a copy of each of the following files:
   - `ndtrack.sh` – This script determines the operating system on which it is running, and installs and then executes the platform-specific version of `ndtrack` (known as "the tracker", the core code element for inventory gathering).
   - `ndtrack.ini` – This is the configuration file for the tracker (on UNIX-like platforms only). This file is optional, and only needed if you wish to configure the behavior or settings for the tracker. If used, it must be installed in the same folder as the tracker. For more information about customizing the configuration, see Configuring ndtrack.ini for UNIX-like Platforms .
   - `InventorySettings.xml` – Functionality extensions are embedded in the file `InventorySettings.xml`, which may be updated from time to time in the ARL download process. Your current license terms are integrated with the same file on the central application server, and the result is automatically downloaded to your inventory beacons after each update. On the inventory beacon, the file is saved in `%CommonAppData%\Flexera Software\Beacon\InventorySettings`. A copy is also saved in the folder identified above for remote execution. The file is mandatory if you wish to collect Oracle inventory (or Microsoft Server inventory, but that is hardly relevant for UNIX-like platforms). If used, this file must also be installed in the same folder as `ndtrack.sh`. It may be omitted for a particular installation of the shell script where this installed instance does not collect Oracle inventory.

     Important: *Do not edit this file!*

     Tip: *Take note of the revision number in the first line of the `InventorySettings.xml` file for future*

reference. Set a reminder for yourself in your preferred tool to check the revision number of `InventorySettings.xml` *after future updates of the Application Recognition Library. If the revision changes, you should manually update all the copies you manually deployed. These changes allow the inventory tool to stay abreast of inventory changes necessary to support the latest Oracle or SQL Server licenses.*

Configuring ndtrack.ini for UNIX-like Platforms

The optional configuration file `ndtrack.ini` can be used:

- To disable specific parts of inventory gathering (although doing this places at risk your ability to calculate consumption for licenses that rely on the inventory being available)
- To store run-time preferences that would otherwise be required on the command line at each invocation.

All the specifications embedded in the default `ndtrack.ini` file are also embedded in the `ndtrack.sh` executable, so that when the `.ini` file is omitted, default functionality is preserved. When the tracker is first invoked, it checks for the presence of an `ndtrack.ini` file in the same directory where the executable is running. If present, the external file takes precedence, *in its entirety*. This means that if, for example, your copy omits a provider statement that is present in the default file, this is equivalent to turning off that part of inventory gathering. For this reason, it is critically important to always start customization with a complete copy of the latest default file from your inventory beacon (see FlexNet Inventory Scanner: Implementation on Unix-Like Platforms), and to change only those elements that are essential, preserving all other values.

Disabling a part of inventory

Find the appropriate section in the `ndtrack.ini` file, and remove or comment out the provider details. For example, the following change prevents collection of the physical memory size on Solaris platforms:

```
[ManageSoft\Tracker\CurrentVersion\SunOS\Hardware\MGS_PhysicalMemory]
#provider=SolarisPhysicalMemory
```

Note: *A line commencing with the hash character (#) is commented out. This character is also known as the pound character, or number sign.*

Be cautious about what you comment out, since an unusual variety of hardware attributes can affect consumption calculations on different kinds of licenses.

Storing a run-time preference

The behavior of the `ndtrack.sh` executable can be conditioned by a large number of preferences. When the parallel executable on Windows, `ndtrack.exe`, is operating within the locally-installed, complete FlexNet inventory agent, it can read these preferences either from the command line or from values saved in the Windows registry (in contrast, the Windows version of the light FlexNet Inventory Scanner does *not* check registry entries, simplifying its deployment and operation). On UNIX-like platforms, there is (of course) no Windows registry, so that preferences are either:

- Read from the command line as parameters, which parameters are common for both Windows and UNIX-like platforms (see FlexNet Inventory Scanner Command Line for details)

- Saved in a configuration file that acts as an alternative storage medium on these non-Windows platforms.

On UNIX-like platforms, the configuration file is different in these two cases:

- When the complete FlexNet inventory agent is locally installed on the target device running a UNIX-like operating system, the file is called `config.ini`.
- When `ndtrack.sh` is being deployed alone as a light inventory scanner, the file is called `ndtrack.ini`.

The naming difference keeps clear their different purpose and differences in content; but these two files have one thing in common — their ability to store preferences for use by `ndtrack.sh` (acting as a 'virtual' registry). Furthermore, this common functionality is achieved in exactly the same way:

- The equivalent of the Windows registry key is listed inside square brackets
- The following lines under each key show the registry value (or values) set under that key.

For example, suppose that you want the collected inventory from your UNIX-like device uploaded to your inventory beacon. One way is to use the following two preferences on the command line (this example shows an IPv4 address for the `UploadLocation`, and IPv6 addresses may also be used if appropriate in your environment):

```
./ndtrack.sh -o Upload=True -o UploadLocation=http://198.51.100.3/ManageSoftRL
```

Another way is useful when you want to deploy the lightweight inventory scanner but by default have it regularly upload inventory. To achieve this, you can customize the `ndtrack.ini` file with the following addition, and simply deploy this into the same directory as the executable `ndtrack.sh` (the `UploadLocation` can alternatively be the fully qualified server name):

```
[ManageSoft\Tracker\CurrentVersion]
Upload=True
UploadLocation=http://FQServerNameOrIP/ManageSoftRL
```

Such customizations require that you know the equivalent registry paths used on Microsoft Windows (as well as the name/value pairs). Many preferences, complete with the relevant registry paths, are documented in the *Preferences* chapter of this document.

For our previous example, that chapter shows the registry path for the computer-based preference `Upload` in this manner:

```
[Registry]\ManageSoft\Tracker\CurrentVersion
```

Here, the placeholder `[Registry]\` stands for one of the following values, as appropriate for the context:

- The registry base path on Windows 32-bit devices
- The registry base path on Windows 64-bit devices
- The `config.ini` file for the full FlexNet inventory agent locally installed on the device
- The `ndtrack.ini` file for the lightweight deployment of `ndtrack.sh` as a stand-alone inventory scanner.

In the case of the lightweight FlexNet Inventory Scanner, the placeholder `[Registry]\` should be read as "add the following path inside square brackets to your `ndtrack.ini` file". This reading, together with the information in the *Value/range* entry in the relevant topics, produces the example shown above, which is good for anonymous authentication on the upload. The standard URL construct can also be used where an account name and password

are required for the upload, although you may prefer this format on a transitory command line rather than in a plain text file:

```
[ManageSoft\Tracker\CurrentVersion]
Upload=True
UploadLocation=http://AccountName:Password@FQServerNameOrIP/ManageSoftRL
```

In a similar manner, you can include in your `ndtrack.ini` file any other preferences for `ndtrack` from this PDF that are relevant to UNIX-like platforms.

Note: *Any preference setting in `ndtrack.ini` is over-ridden by a different value for the same preference given as a command-line parameter. The priority order is: default (built in) values are over-ridden by settings in `ndtrack.ini`, which are over-ridden by command-line parameters.*

Customizing Searches for FlexNet Inventory Scanner

You can extensively customize the FlexNet Inventory Scanner behavior using the command line by instructing it to include or exclude any of the following:

- Folders (and optionally, their subfolders)
- Particular file extensions
- Specific filenames
- Specific MD5 digest values.

Tip: *Checking MD5 digests across all inventory can significantly extend the time required for gathering inventory on a device.*

When including or excluding extensions, file names and MD5 values, there is a possibility that a file could be both included and excluded. To overcome this problem, a matching MD5 value overrides a file name, which overrides a file extension.

For example, if

- File extension `exe` is included
- Filename `xcopy.exe` is excluded
- MD5 value `123456...` (the MD5 for `xcopy.exe`) is included

then the FlexNet Inventory Scanner includes all files with extension `exe` except for all versions of `xcopy.exe` that do not have an MD5 value `123456...`

If the same directory, file extension, filename or MD5 value is explicitly included and excluded, the exclusion command takes precedence.

All such customizations are available through command-line options when you execute the FlexNet Inventory Scanner (see FlexNet Inventory Scanner Command Line). On UNIX-like platforms only, in addition to the command-line

options, they may be configured in the `ndtrack.ini` configuration file (see Configuring ndtrack.ini for UNIX-like Platforms).

FlexNet Inventory Scanner Command Line

Command line summary for the FlexNet Inventory Scanner on both Windows and UNIX-like platforms.

Syntax:
(On Microsoft Windows) **`FlexeraInventoryScanner.exe [options...]`**

Syntax:
(On UNIX-like platforms) **`ndtrack.sh [options...]`**

Options:

Syntax:
`-o` *tag* = *value*

These optional parameters individually override the default FlexNet Inventory Scanner settings on Windows. On UNIX-like platforms, they override both the individual default settings and any matching preference recorded in `ndtrack.ini`.

Preferences for FlexNet Inventory Scanner are directly passed through to the `ndtrack` executable, so that details for each option are included in the preferences topics listed and linked below.

Enclose values in double quotation marks if the values include spaces; otherwise, double quotation marks are optional. Special characters (double quotation marks, backslash) must be escaped with a backslash.

Tip: *The `-t` command-line option available on Windows for the installed FlexNet inventory agent is not supported for the FlexNet Inventory Scanner. Instead, you can specify the inventory type using the command line parameter:*

```
-o InventoryType=User|Machine
```

If the `InventoryType` preference is not specified, the type of inventory collected on Windows platforms depends on the account running the FlexNet Inventory Scanner:

- *For the **LocalSystem** account, a computer (machine) inventory is collected*
- *For all other accounts, a user inventory is collected.*

For UNIX-like platforms, only machine-based inventory is supported.

Return codes

FlexNet Inventory Scanner returns a zero on success. If you receive a non-zero return code, check the log file. Details of the log file may also be configured with command-line options, as listed below.

Example: Command line examples
This example collects a computer inventory and stores it locally (on the computer device where the FlexNet

Inventory Scanner is executing) for upload by a separate system:

```
FlexeraInventoryScanner.exe
          -o InventoryType=Machine
          -o MachineZeroTouchDirectory="local-folder"
          -o Upload=False
```

This example collects user-based inventory and uploads it to an inventory beacon:

```
FlexeraInventoryScanner.exe
          -o InventoryType=User
          -o UploadLocation="http://InventoryBeacon/ManageSoftRL"
```

Options

Except as specifically noted, the same options are supported for the FlexNet Inventory Scanner on Microsoft Windows, and for the use of `ndtrack.sh` as a scanner on UNIX-like platforms.

- DateTimeFormat
- ComputerDomain
- ExcludeDirectory
- ExcludeExtension
- ExcludeFile
- ExcludeMD5
- GenerateMD5
- Hardware
- IncludeDirectory

  Important: *Since the default for this preference is blank, this means that no files are included in inventory gathering by default. Inventory then relies entirely on installer evidence. If you wish to collect file evidence for any reason, it is mandatory to set an appropriate value for* `IncludeDirectory`*.*

- IncludeExecutables
- IncludeExtension
- IncludeFile
- IncludeMachineInventory
- IncludeMD5
- IncludeRegistryKey (ignored for UNIX-like platforms)
- IncludeUserInventory (ignored for UNIX-like platforms)

- InventoryFile
- InventoryScriptsDir
- `InventoryType` is deprecated on Windows and ignored on UNIX-like platforms. The defaults for Windows are `User` unless the account executing FlexNet Inventory Scanner is `LocalSystem`, in which case the default switches to `Machine`. Available for backward compatibility only.
- LogFile (installation component)
- LogLevel (inventory component)
- LogModules (inventory component)
- LowProfile (inventory component)
- MachineName
- MachineZeroTouchDirectory (ignored for UNIX-like platforms)
- MSI (ignored for UNIX-like platforms)
- PreferIPVersion
- ProgramFiles, ProgramFilesX86Folder, ProgramFilesX64Folder (ignored for UNIX-like platforms)
- Recurse
- RunInventoryScripts (ignored for UNIX-like platforms)
- ShowIcon (inventory component) (ignored for UNIX-like platforms)
- SysDirectory (ignored for UNIX-like platforms)
- Upload

Tip: *The default `False` value for FlexNet Inventory Scanner is the inverse of the default for the installed, complete FlexNet inventory agent.*

- UploadLocation
- `UserHardware` (ignored for UNIX-like platforms, and deprecated for Windows). Available for backward compatibility only. Effective only when running in the user context (also deprecated). The default `False` excludes hardware inventory data from the user's software inventory.
- `UserZeroTouchDirectory` (ignored for UNIX-like platforms, and deprecated for Windows). Available for backward compatibility only. The FlexNet Inventory Scanner uses the location specified in this option for any user-based inventory (user-based inventory is also deprecated). It has no default value.
- VersionInfo (ignored for UNIX-like platforms)
- WinDirectory (ignored for UNIX-like platforms)
- WMI (ignored for UNIX-like platforms)
- WMIConfigFile (ignored for UNIX-like platforms).

Notes

1. Default values only apply if the parameter is not specified.
2. Directory paths must be specified as absolute paths (that is, on Windows, starting with a drive name). The typical wildcards in directory names are supported (`*` representing any number of characters, and `?` representing a single character).
3. The FlexNet Inventory Scanner accepts all name/value combinations, although if a preference is used that does not appear in the list above, it may be ignored. On UNIX-like platforms, preferences may be included in the `ndtrack.ini` configuration file (not supported for Microsoft Windows).
4. A preference value can symbolically refer to another preference by enclosing its name thus: `$(`*`preferenceName`*`)`. References can contain further references.
   Example: The command

   ```
   FlexeraInventoryScanner.exe -o IncludeDirectory=$(WinDirectory)
   ```

   includes the Windows directory in the scan. References are resolved after all preferences are loaded so there are no ordering issues. Hopefully it is self-evident that, on UNIX-like platforms, only supported preferences can be referenced.
5. Semicolon or comma-separated values are the only method for defining multiple values in the FlexNet Inventory Scanner. Only the `Include` and `Exclude` preferences listed above may have multiple values. All other preferences contain a single value that can be overwritten.
   Example: The command

   ```
   FlexeraInventoryScanner.exe -o IncludeDirectory=C:\\;D:\\;E:\\
   ```

   will scan the computer's C:, D:, and E: drives if they are fixed (hard) disks, but not if they are CD-ROM drives or logical drives (mapped to network locations).

For more information, see ndtrack Command Line.

FlexNet Inventory Scanner: Troubleshooting Inventory

Inventory gathering and upload is a sophisticated chain from target inventory device through inventory beacon to central application server. For general trouble-shooting over the whole process, see the online help for FlexNet Manager Suite under *Inventory Beacons* > *Inventory Beacon Reference* > *Troubleshooting: Inventory Not Uploading*. This topic focuses entirely on inventory collection on the target inventory device.

When you use FlexNet Inventory Scanner, the normal log file for the `ndtrack` executable is saved:

- On Windows platforms, in `%temp%\ManageSoft\tracker.log`
- On UNIX-like platforms, in `/var/tmp/flexera/log` (when the executable was invoked by the `root` account, as recommended) or `/var/tmp/flexera.`*`UserName`*`/log` (when invoked by the *`UserName`* account).

For advanced trouble-shooting, you may require more advanced tracing and logging. You may also be asked to submit a trace file to assist the Support team at Flexera to solve difficult problems in your environment.

By default, tracing is not a function deployed with FlexNet Inventory Scanner. However, with some custom preparation, you can set up for, and control, tracing with a `.trace` configuration file of your own.

To set up and configure tracing for FlexNet Inventory Scanner:

1. Obtain a copy of an `etcp.trace` file from an installed FlexNet inventory agent on another device.

   Where the full FlexNet inventory agent has been installed on a target device, the `etcp.trace` file is located with the `ndtrack` executable:

   - On Windows, the default is `C:\Program Files (x86)\ManageSoft\etcp.trace`
   - On UNIX-like platforms, the default is `/opt/managesoft/etcp.trace`.

   Because this file format is consistent across platforms, you may take your copy of `etcp.trace` from any inventory device where the full FlexNet inventory agent is installed.

2. On a Windows device:

   a. Save the `etcp.trace` file in the directory on the target device where the FlexNet Inventory Scanner will execute (the `temp` directory of the account that is invoking it).

   b. On the same target device, create the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ManageSoft Corp\ManageSoft`, add the string value `InstallDir`, and set the value to the full path to the directory where you have saved `etcp.trace`.

3. On UNIX-like platforms, a rename and relocation are mandatory:

   a. Rename the `etcp.trace` file as `ndtrack.trace`.

   b. Save the file as `/etc/ndtrack.trace`.

4. Configure the name and location of the trace/log file that will be generated on the inventory device.

   The hash or pound character (#) identifies a comment. To "uncomment" a line in the `.trace` configuration file means to delete (only) the leading hash character. Choose one of the following lines, uncomment it, and optionally modify it to your requirements. On Windows:

   ```
   #filename=C:\ManageSoft.log
   #filename=C:\ManageSoft%p_%d_%t_%u.log        # filename pattern with everything!
   ```

   On UNIX-like platforms:

   ```
   #filename=/tmp/log/mgstrace.log
   #filename=/tmp/log/ManageSoft%p_%d_%t_%u.log # filename pattern with everything!
   ```

   See the notes within the file header for the use of the supported variables within the file name.

Tip: *It is best practice to use a pattern that includes (at least) either a date stamp (%d) or a sequential number (%u). Without these, the fixed file name means tracing information is appended to the same file with every inventory collection. This can quickly produce a trace file too large for text editors to read, and too hard to manage in terms of disk space. Variables in the file name trigger creation of a new file each time the associated variable is changed (or, for %u, at every invocation of `ndtrack`).*

Important: *The log file path:*

- *Must be on the same drive as the* `ndtrack` *executable (on Windows devices)*
- *Must exist and be writable before the* `ndtrack` *executable is next invoked (tracing does not create any directories, and does not function if any directory in the specified path is missing or unwritable).*

5. Uncomment the lines for which you want to enable tracing (ensuring that the uncommented line now starts with a plus sign).

   For FlexNet Inventory Scanner, the typical lines to uncomment are:

```
+Inventory
+Error
+Communication/Network
```

When FlexNet Inventory Scanner is invoked, it creates the tracing log file as you specified, ready for your inspection.

6

Core Deployment: Details

This chapter provides great detail about the FlexNet inventory core components when you deploy them to target Windows devices using the third-party (or non-FlexNet) deployment technology of your choice.

Even though the same components can be invoked, this approach is a quite different method from the FlexNet Inventory Scanner approach:

FlexNet inventory core components	FlexNet Inventory Scanner
You deploy and install the components you need (presumably only once per version).	A self-installing executable installs the tracker (or, on UNIX-like platforms, a shell script installs the correct version of the tracker for the target operating system). This installation process is repeated for every invocation of the FlexNet Inventory Scanner.
You schedule invocation of the components with your preferred scheduling tool.	The tracker is invoked automatically immediately upon installation.
After execution, the components remain in place awaiting the next invocation.	After execution, the tracker is automatically uninstalled. The FlexNet Inventory Scanner remains available to repeat the cycle when required.

In this configuration, and given appropriate command lines, the FlexNet inventory core components are capable of collecting hardware and software inventory from a target device, and uploading it to a location specified in the command line. This configuration is helpful when you want to take full control of deployment, scheduling, agent updates, and the like, using your existing processes and technologies. (For details of the distinct use cases, refer back to Understanding What, Where, How, and Why.)

This document provides a consistent set of data (as far as possible) across all the different use cases, each in its own chapter. This means that, once you have chosen your preferred use case, you can focus only on the details for that one, and ignore all other use case chapters.

In addition to the distinct chapters for the different use cases, you should also review the subsequent chapter on functionality that is common throughout. This is followed by detailed reference material on command lines, preferences, file formats, and the like.

Core Deployment: Normal Operation

For details about deployment and configuration of the FlexNet inventory core components, see Core Deployment: Implementation. To help you decide whether to proceed with that process, this topic describes the resulting operation in some detail, assuming that you have deployed the FlexNet inventory core components into locations within your Windows computer estate where each installation can access one or more inventory beacons, and function normally. Furthermore, you have configured a Microsoft scheduled task to invoke the tracker component (`ndtrack.exe`) on each inventory device, setting the appropriate command line parameters.

The entire process is covered, including what happens to the collected data after it uploaded by the FlexNet inventory core components. Each numbered step provides a summary point, followed by further specific details that you can skip over until needed.

Tip: *Inventory operations of the installed FlexNet inventory core components in the Core deployment case are never controlled by inventory rules created in the web interface of FlexNet Manager Suite. The results of those settings are distributed through policy, and the FlexNet inventory core components do not include any mechanism for requesting, downloading, or applying policy. To have the target inventory device managed through settings in the web interface, switch instead to either of the Adopted case or the Agent third-party deployment case.*

1. The FlexNet inventory core components sit dormant on the target inventory device until your custom scheduled task triggers the `ndtrack.exe` component to collect inventory.
2. In accordance with the command-line parameters included in the invocation, the tracker collects the hardware and software inventory from its local device.

   By default, the tracker runs at low priority so that higher priority tasks are not interrupted, and there is minimal impact on system performance. Inventory details are saved in two ways:

   - An `.ndi` file is saved (by default) in `$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Tracker\Inventories`. This directory preserves the local copy of the inventory file (where you can inspect its contents) until it is over-written at the next inventory collection by the same account (since inventory file naming reflects the account running the inventory collection).
   - At the same time, a compressed (`.ndi.gz`) copy of the file is also saved, ready for upload, by default in `$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Common\Uploads\Inventories`. (If this file is subsequently uploaded successfully, it is removed; but if there is a failure it remains in this directory until over-written at the next inventory collection by the same account.)
3. If the tracker has been configured for Oracle inventory (because you also deployed the `InventorySettings.xml` file), and has uncovered any Oracle services running on the local device, and the tracker is running as `root`, additional files are created:

   - A second `.ndi.gz` file of Oracle inventory is also generated, and saved in `$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Common\Uploads\Inventories` (an uncompressed version is not saved).
   - As well, the Oracle discovery is reported in an uncompressed `.disco` file, saved in the `$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Common\Uploads\Discovery` directory.

   Note: *On UNIX-like platforms, Oracle discovery and inventory gathering is only possible when the tracker*

(ndtrack executable) is running as root.

4. After collecting the hardware and software inventory, ndtrack immediately attempts to transfer the compressed inventory file(s) to an inventory beacon.

   - The inventory beacon is the one identified in the command-line parameters when the tracker was invoked (either of the UploadLocation or UploadSettings options may have been used).
   - If the upload succeeds, the compressed inventory files are removed from $(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Common\Uploads\Inventories.
   - Results are logged to $(TempDirectory)\ManageSoft\tracker.log by default (although there are other command line options that can modify this).
   - The upload is a background process that does not take priority away from other current tasks running on the inventory device.
   - If the initial upload is unsuccessful for any reason, there is no attempt at a catch-up upload overnight (that functionality requires the full FlexNet inventory agent). Either you may script a catch-up, or the compressed files are left in place, and are overwritten at the next inventory collection trigger using the same account name.

5. FlexNet Beacon (the code entity on the inventory beacon) uploads the inventory data to its parent on a schedule set by the Microsoft Scheduled Task Upload FlexNet logs and inventories (by default, repeating every minute throughout the day).

   The checking cycle when the folder is empty is very quick and does not perceptibly load the inventory beacon, even though it is frequently repeated. The parent of an inventory beacon may be the central application server, or another inventory beacon if these have been arranged in a hierarchy. In the latter case, each inventory beacon in turn repeats the upload process until the data reaches the application server.

6. On the application server (or, in a scaled-up system with separate servers, the inventory server), the web service ManageSoftRL receives the uploaded packages for both inventory and (if configured) usage tracking (and other uploaded files).

   These are processed immediately, being loaded into the internal operations databases: inventory (.ndi) and usage (.mmi) files are loaded into the inventory database; any Oracle discovery (.disco) file is loaded into the compliance database. If the service gets overloaded, it will temporarily spool incoming files to its local *%CommonAppData%*\Flexera Software\Incoming\Inventories directory (or the peer Discovery folder for any Oracle discovery file). From these folders, file import is resumed under the control of Microsoft scheduled tasks (for example, Import inventories, which is triggered every 10 minutes).

7. On the next inventory import and license consumption calculation, the inventory and usage data is collected from the inventory database, socialized as necessary, and imported into the compliance database. Here it is used in license calculations, and made available in management views and reports.

   This import step can be triggered in one of three ways:

   - Normally, the batch scheduler triggers an import daily (by default, at 2am local time on your application server), with the license consumption calculation triggered thereafter. This default time is configurable by editing the Microsoft scheduled task Inventory import and license reconcile on your application server (or, in larger implementations, batch server).

- An operator in the `Administrator` role can choose to import the waiting inventory and trigger license consumption calculation, or reconciliation, as soon as possible (navigate to **License Compliance > Reconcile**).
- For testing, a knowledgeable system administrator could use a command line on your application server (or, in a scaled-up system, your batch server) like:

  ```
  BatchProcessTask.exe run InventoryImport
  ```

  (for details, see the *Server Scheduling* chapter in *FlexNet Manager Suite System Reference*).

Core Deployment: System Requirements

The following details apply to the FlexNet inventory core components when you use your preferred third-party deployment processes to install them on a target Windows device.

Supported platforms

The FlexNet inventory core components operate on the following platforms (inventory targets):

Microsoft Windows	UNIX-like platforms

• Windows Server 2008 R2 x64, 2012, 2012 R2, 2012 R2 SP1, 2016, 2019, 2022
• Windows Server Core 2008, 2008 R2 x64, 2012, 2012 R2
• Windows Server Standard (previously known as Windows Server Core) 2016, 2019
• Windows Vista, 7, 8, 10, 11
 	
 Note: *Windows systems that can run on ARM-based devices are also supported.*
	• AIX 7.1 LPARs with Technology Level 5 or later, AIX 7.2
	• Amazon Linux 2, 2023 (ARM64/AArch64; x86, 32-bit and 64-bit)
	• CentOS 6-7.5 (x86, 32-bit and 64-bit); 7.6-9 (x86 64-bit only)
	• Debian Linux 8–11.3 (x86, 32-bit and 64-bit); 11.5, 12.0 (x86 64-bit only)
	
	 Note: *For Debian Linux 9.4, 9.5 and 10 (both 32-bit and 64-bit), minimal installations of the OS core omit the `ifconfig` command. This prevents collection of the IP address and MAC address in inventory. The `root` user can use the following command to restore this functionality:*
	`apt-get install net-tools -y`
	• Fedora Linux 26 (x86, 32-bit and 64-bit); 27-38 (x86 64-bit only)
	• HP-UX 11i v3 (Itanium only), vPars/nPars
	• macOS 10.15.4–14 (applicable for both Intel and Apple M-series processors)
	
	 Note: *If you need to run the FlexNet inventory agent of version 20.1 or earlier on an Apple M-series processor ("Apple silicon"), Rosetta 2 must be installed and running. This is Apple's solution for transitioning most Intel-based applications to run on Apple silicon. There are two possible command formats for installing Rosetta 2:*
	◦ *Interactive installation that asks for agreement to the Rosetta 2 license:*
	`/usr/sbin/softwareupdate --install-rosetta`
	◦ *Non-interactive installation:*
	`/usr/sbin/softwareupdate --install-rosetta --agree-to-license`

Microsoft Windows	UNIX-like platforms
	• Nutanix AHV hypervisor ***Note:*** *All versions that are within their product support lifecycle and use the* `libvirt` *library are supported.* • OpenSUSE Leap 42.2, 42.3 (x86, 32-bit and 64-bit); 15-15.4 (x86 64-bit only) • Oracle Linux 6.0–6.10 (x86, 32-bit and 64-bit); 7.0-8.7 and 9.0-9.1 (x86 64-bit only) • Photon OS 3.0-5.0 • Red Hat Enterprise Linux (RHEL) 6.0-6.10 (x86, 32-bit and 64-bit); 7.0-8.8 and 9.0-9.2 (x86 64-bit only) • Rocky Linux 8, 9 • Solaris 10–11 (SPARC), Zones for versions 10–11 • Solaris 10–11.4 (x86), Zones for versions 10–11 • SUSE Linux Enterprise Server 12 SP3, 12 SP4, 12 SP5, 15, 15 SP1, 15.2-15.5 (x86 64-bit only) • Ubuntu 14–17.04 (x86, 32-bit and 64-bit); 17.10-23.04 (x86 64-bit only) Linux on IBM zSystems platforms are also supported. The following Linux distributions are certified for usage on the IBM zSystems architecture. For details about supported versions on different hardware types, see https://www.ibm.com/support/pages/linux-ibm-z-tested-platforms. • Red Hat Enterprise Linux • SUSE Linux Enterprise Server • Ubuntu

Disk space requirements

On target Windows inventory devices in the Core deployment case, the following are disk space requirements:

- Where the target device is a typical workstation, in the order of 6MB
- For an Oracle server, in the order of 9MB.

The following log file is available on the target inventory device in the Core deployment case:

- `tracker.log` — Generated by the inventory component, `ndtrack`

Memory requirements

On target inventory devices, the memory requirements are:

- Minimum RAM: 512 MB
- Recommended RAM: 2 GB

In general, through a cycle of inventory gathering and upload, the memory demand is in the order of 5-30 MB.

Communications protocols and ports

When the FlexNet inventory core components execute on the target device (in the Core deployment case), the only ports required are the standard ports for communication with the inventory beacon:

- From FlexNet inventory core components on target device, inbound on the inventory beacon — File upload using HTTP protocol: port 80
- From FlexNet inventory core components on target device, inbound on the inventory beacon — File upload using HTTPS protocol: port 443

Supported packages to inventory

FlexNet inventory can include data from most package technologies supported by the operating systems, and some additional third-party packaging technologies:

Platform	Supported package technologies
All platforms	InstallAnywhere (IA), InstallShield Multiplatform (ISMP), BEA/Oracle Installer (BEA), Oracle Universal Installer (OUI), IBM Installation Manager (IIM)
AIX	LPP, RPM
HP-UX	Software Distributor SD-UX Package
Linux	RPM (Red Hat, CentOS, Oracle, SuSE, Fedora, etc), DPKG (Debian, Ubuntu).
Mac OS X	Mac Application Bundle, Mac Package Bundle
Solaris	Sys V Package (pkg), IPS
Windows	MSI, Add/Remove Programs Registry Key

However, FlexNet inventory cannot collect data from some of the less common or newer operating system technologies and many third-party technologies. Some known examples include:

- All platforms — IBM InstallStream, IBM Tivoli Netcool Installer
- Mac OS X — Mac flat package.

System load benchmarks

The following notes reflect observed behavior on sample target inventory devices during collection of FlexNet inventory:

Task	Run duration (seconds)	CPU usage (seconds)	CPU usage (% of single core)	Memory usage	Network load
Inventory collection	13 to 240 s	5 to 130 s	10% to 50%	4 MB to 20 MB	10 KB to 200 KB per upload

Core Deployment: Accounts and Privileges

In the Core deployment case, when the FlexNet inventory core components gather hardware and software inventory from the target Windows inventory device, they execute as the user name (or account) that triggered the execution.

To effectively gather inventory, this account must have administrator privileges on the local machine. The preferred practice is for the FlexNet inventory core components to be invoked by the Local `SYSTEM` user. In production use, this can be configured when you set up the Microsoft scheduled task that is to trigger inventory collection.

Core Deployment: Implementation

Deploying on the FlexNet inventory core components, and doing so with third-party tools, is not the recommended best practice for using FlexNet Manager Suite to gather specialized inventory. It is a path available for those who cannot use any of the other preferred paths (in particular, third-party deployment of the complete FlexNet inventory agent in the Agent third-party deployment case offers much better automation and simplified on-going management). Because it is not recommended, there is no easy path to securing and configuring all the required elements.

In particular, separate executables are not readily available for the various UNIX-like platforms, so that only a path for Windows platforms is described here. (On UNIX-like platforms, the closest equivalent is to use the FlexNet Inventory Scanner, which takes care of platform-specific differences. For details, see FlexNet Inventory Scanner: Implementation on Unix-Like Platforms.)

To use third-party deployment tools for the FlexNet inventory core components on Windows platforms:

1. Obtain a copy of the appropriate files from a convenient inventory beacon:
   a. In Windows Explorer on your inventory beacon, navigate to `C:\Program Files (x86)\Flexera Software\Inventory Beacon\RemoteExecution\Public\Inventory`.
   b. Take a working copy of all files in this directory (including the `plugins` subdirectory), with the exception of the two files for UNIX-like platforms (`ndtrack.sh` and `ndtrack.ini`).
   c. Save this collection to a working location suitable for preparing deployment with your preferred technology.

   These files are the version of the tracker component that is automatically installed with the FlexNet Beacon software. You may wish to note in your procedural documentation that a future update to your inventory beacons should also trigger a re-deployment of the updated tracker.

2. Optionally, in that working location, customize the `wmitrack.ini` file to reconfigure the WMI classes used for inventory collection on target Windows devices.

   The components to be tracked can be any valid Win32 classes. A full list of classes is provided at https://msdn.microsoft.com/en-us/library/aa394583%28v=vs.85%29.aspx.

3. Optionally (and subject to your license terms), extend the inventory gathering capabilities to cover Oracle inventory, or Microsoft SQL Server inventory, on target devices:

   This functionality is only available if you have licensed the FlexNet Manager for Datacenters product.
   Check that your saved folder contains a copy of `InventorySettings.xml`.

   Important: *Do not edit this file!*

   Tip: *Take note of the revision number in the first line of the `InventorySettings.xml` file for future reference. Set a reminder for yourself in your preferred tool to check the revision number of `InventorySettings.xml` after future updates of the Application Recognition Library. If the revision changes, you should manually update all the copies you manually deployed. These changes allow the inventory tool to stay abreast of inventory changes necessary to support the latest Oracle or SQL Server licenses.*

   For this Core deployment case, the `InventorySettings.xml` file must be deployed to the same directory as the `ndtrack` executable. (Don't be confused by other cases, such as the Agent third-party deployment case, where the `InventorySettings.xml` file must *not* be co-located with the full agent.)

4. Configure your deployment tool to set a schedule for inventory collection and upload on each target inventory device (for example, using a Microsoft Scheduled Task, or your preferred scheduling technology).

   - The task should run under an account with administrator privileges, preferably the local `SYSTEM` account. When run as `SYSTEM`, it is not necessary to specify the inventory type, as the default for this account is to take machine inventory. If you run under any other account, you *must* include the `-t Machine` command line parameter.
   - The command line *must* include a parameter to attempt uploads to an inventory beacon (preferably a high-reliability server and network, since there is no catch-up from failed upload attempts), such as the following (on a single line, and inserting your inventory beacon server's domain name in place of the place-holder):

     ```
     drive-and-path\ndtrack.exe -t Machine
         -o UploadLocation="http://InventoryBeacon/ManageSoftRL"
     ```

     See UploadLocation for alternative formats, including the use of IPv4 or IPv6 addresses.
   - The command line may include other parameters discussed in ndtrack Command Line.
   - You may need to vary (or randomize) the scheduled time for inventory collection, to spread the load on both the network and the inventory beacon accepting the subsequent uploads.

5. As required for your deployment tool/method, pack and deploy the modified folder of files.

   The typical folder for installation of the FlexNet inventory core components is:

   ```
   C:\Program Files (x86)\Flexera\Agent
   ```

However, you may customize the installation path as required (for example, to install on another fixed disk drive).

6. Validate by visiting a targeted Windows inventory device, and using the scheduling tool to trigger an immediate (one-off) inventory collection and upload.

   Alternatively, use the same command line and options in the command-line window. For details of saved files, paths, and logs, see Core Deployment: Normal Operation.

The installed FlexNet inventory core components continue to collect and upload hardware and software inventory (and, where applicable, additional Oracle and Microsoft SQL Server inventory and discovery data) as scheduled. Because there is no download of policy possible to the FlexNet inventory core components, inventory gathering on these devices is not controlled by any aspects of the web interface of FlexNet Manager Suite. Control, and future management of updates, are entirely in your hands.

Core Deployment: Troubleshooting Inventory

Inventory gathering and upload is a sophisticated chain from target inventory device through inventory beacon to central application server. For general trouble-shooting over the whole process, see the online help for FlexNet Manager Suite under *Inventory Beacons* > *Inventory Beacon Reference* > *Troubleshooting: Inventory Not Uploading*. This topic focuses entirely on inventory collection on the target inventory device.

After you have deployed and installed the FlexNet inventory core components on a Windows device, the regular log file for the `ndtrack` component is `$(TempDirectory)\ManageSoft\tracker.log` by default (although there are command line options that can modify this).

For advanced trouble-shooting, you may require more advanced tracing and logging. You may also be asked to submit a trace file to assist the Support team at Flexera to solve difficult problems in your environment.

To set up and configure tracing for FlexNet inventory core components:

1. Obtain a copy of an `etcp.trace` file from an installed FlexNet inventory agent on another device.

   Where the full FlexNet inventory agent has been installed on a target device, the `etcp.trace` file is located with the `ndtrack` executable:

   - On Windows, the default is `C:\Program Files (x86)\ManageSoft\etcp.trace`
   - On UNIX-like platforms, the default is `/opt/managesoft/etcp.trace`.

   Because this file format is consistent across platforms, you may take your copy of `etcp.trace` from any inventory device where the full FlexNet inventory agent is installed.

2. Save the `etcp.trace` file in the directory on the target device where the FlexNet inventory core components will execute.

3. Create a registry key that identifies this location:

   a. Create the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ManageSoft Corp\ManageSoft`.

   b. Add the string value `InstallDir`.

c. Set the value to the full path to the directory where you have saved `etcp.trace`.

4. Configure the name and location of the trace/log file that will be generated on the inventory device.

   The hash or pound character (#) identifies a comment. To "uncomment" a line in the `.trace` configuration file means to delete (only) the leading hash character. Choose one of the following lines, uncomment it, and optionally modify it to your requirements.

   ```
   #filename=C:\ManageSoft.log
   #filename=C:\ManageSoft%p_%d_%t_%u.log        # filename pattern with everything!
   ```

   See the notes within the file header for the use of the supported variables within the file name.

   Tip: *It is best practice to use a pattern that includes (at least) either a date stamp (%d) or a sequential number (%u). Without these, the fixed file name means tracing information is appended to the same file with every inventory collection. This can quickly produce a trace file too large for text editors to read, and too hard to manage in terms of disk space. Variables in the file name trigger creation of a new file each time the associated variable is changed (or, for %u, at every invocation of `ndtrack`).*

   Important: *The log file path:*

   - *Must be on the same drive as the `ndtrack` executable (on Windows devices)*
   - *Must exist and be writable before the `ndtrack` executable is next invoked (tracing does not create any directories, and does not function if any directory in the specified path is missing or unwritable).*

5. Uncomment the lines for which you want to enable tracing (ensuring that the uncommented line now starts with a plus sign).

   For FlexNet inventory core components, the typical lines to uncomment are:

   ```
   +Inventory
   +Error
   +Communication/Network
   ```

When the tracker component is invoked, it creates the tracing log file as you specified, ready for your inspection.

7

Common: Details

This chapter provides useful details that apply to the operation of the `ndtrack` executable, regardless of whether that is deployed as part of FlexNet inventory agent or of FlexNet inventory core components. The details in this chapter are also unaffected by the deployment methods.

Earlier chapters provide a consistent set of data (as far as possible) across all the different deployment methods, each in its own chapter. This means that, once you have chosen your preferred deployment method, you can focus only on the details for that one, and ignore all other deployment method chapters.

In contrast, this chapter focuses on functionality that is common throughout.

Common: Child Processes Invoked by the Tracker

Whenever the tracker (the `ndtrack` executable) is invoked on a target device, it calls a few utilities and operating system commands to fulfill its inventory-gathering tasks. The behavior of the tracker is consistent (per platform), regardless of how it is delivered to the target device or how it is invoked. For this reason, the following per-platform listings of child processes invoked by the tracker apply equally across all these tracker cases described in this document:

- Adopted through the automated processes within FlexNet Manager Suite
- Agent third-party deployment using the third-party tools of your choice
- Zero-footprint, where the tracker is copied from the inventory beacon, and installed and run
- FlexNet Inventory Scanner, where the tracker is invoked by the `ndtrack.sh` script
- Core deployment, where you have arranged for the deployment of FlexNet inventory core components and now manage invoking the `ndtrack` executable within your specialized scenario

(To revise details of the cases, see the chapter Understanding What, Where, How, and Why.)

To operate securely, the tracker invokes these child processes with appropriate levels of privilege. Security and functionality can often be a trade-off, and in those cases, the priority is with security. This may mean that, depending on the security configuration in your computing estate, some inventory-gathering processes cannot succeed until you

update the configuration to allow the specific tasks. In many cases, the outcomes of such restrictions are visible in the discovered device properties by selecting the **Status** tab, and expanding the **Oracle database inventory** heading. However, some issues are recorded only in the `tracker.log` file on the target inventory device. For more details, see the chapter on *Oracle Discovery and Inventory* in the *FlexNet Manager Suite System Reference* PDF.

Because the details vary across platforms, the processes and accounts used are documented separately in the following topics, first for Windows and then for UNIX-like platforms. Each gives a complete listing, and they may be read independently.

Common: Child Processes on UNIX-Like Platforms

The tracker normally runs as `root`, because elevated privileges are required to complete several aspects of inventory gathering.

Note: *If you choose to run the tracker using another account that does not have elevated privileges, you considerably weaken the resulting inventory:*

- *Oracle inventory is disabled*
- *IBM WebSphere inventory is disabled*
- *Inventory from IBM Db2 Database and optional add-ons is disabled*
- *All hard disk information for Linux systems is excluded*
- *Software inventory from paths not accessible to the executing user is omitted for all systems*
- *Several further losses occur, as noted in the table of child processes below.*

The implications of running as `root` include the following:

- Commands in safe system paths (not writable by other users) are run as `root`.
- Commands found within paths listed in the `$PATH` environment variable for the `root` user are run as `root`.

  Note: *This makes it important that, as is normal secure practice, you do not allow any unsecured directories to be included in the $PATH environment variable for the `root` user.*

- Commands and utilities saved in unsecured directories on the file system are *not* run as `root`. These must be run with no more trust that you already provide in your environment. To do this, the tracker uses *user impersonation*, so that it invokes child processes with the same level of trust and security management that you have already established for the existing account being impersonated. On UNIX-like platforms, the method is to impersonate the user account that is running the service related to the executable in question. For example, the executable `lsnrctl` normally starts the `tnslsnr` service. Therefore, when the tracker needs to invoke `lsnrctl`, it impersonates the user account running the `tnslsnr` service. Since this account is already running the process in question, it is a trusted account for the path on the target device where inventory is being collected.

The table of child processes is organized in alphabetical order of the executables invoked by the tracker. Where the details vary across various UNIX-like platforms, a separate entry exists for each [group of] platform[s] where the command line is distinct.

Tip: *The* `date` *command is not in the following list because it is not invoked by the tracker. It is invoked in the Zero-footprint case when the remote inventory beacon tests to see whether the account (recovered from its Password Manager) can successfully elevate privileges on the target device, in order to complete the process as described in* *Zero-Footprint: Normal Operation**.*

Executable	Platform	Path	Notes
`arp`	All	The following are searched in this order: • `$PATH` • `/usr/sbin`.	*Command line:* `/successfulPath/arp IPaddress` *Purpose:* Reports the MAC address of network interface(s). *Invoked using:* The account running the `ndtrack` executable (default: `root`).
`db2ilist`	Linux, Solaris, AIX	Path returned by `db2ls`	*Command line:* `/successfulPath/bin/db2ilist` *Purpose:* Lists all the database instances running in the context where `db2ilist` is executed (normally, instances from the same database installation that provides the `db2ilist` command). *Invoked using:* The account running the `ndtrack` executable (which in this case must be `root`).
`db2licm`	Linux, Solaris, AIX	Path returned by `db2ls`	*Command line:* `/successfulPath/adm/db2licm -l / -g filename` *Purpose:* Reports inventory of the Db2 product in the *`successfulPath`* (including its product identifier) and its optional add-ons, including the available license information. Once the temporary file (*`filename`*) has been processed, it is deleted. *Invoked using:* The account running the `ndtrack` executable, which in this case must be `root` (otherwise inventory collection for IBM Db2 and add-ons is automatically disabled).

Executable	Platform	Path	Notes
db2ls	Linux, Solaris, AIX	/usr/local/bin	*Command line:* `/usr/local/bin/db2ls -c` *Purpose:* Identifies the path to the IBM Db2 Database installation. *Invoked using:* The account running the ndtrack executable, which for IBM Db2 inventory must be root (otherwise inventory collection for IBM Db2 and add-ons is automatically disabled).
dmidecode	Linux, Solaris (Intel)	The following are searched in this order: • /usr/sbin • /opt/managesoft/libexec • $PATH.	*Command line:* `/successfulPath/dmidecode` *Purpose:* Reports serial number, UUID, manufacturer, model, and chassis type, extracted from the computer's DMI (or SMBIOS) table. ***Tip:*** *On older versions of Linux where this utility is unavailable, an equivalent mgsdmidecode supplied with the full FlexNet inventory agent may be used instead. (This is also run as root.)* *Invoked using:* The account running the ndtrack executable, which in this case must be root (otherwise the relevant elements are missed from the uploaded inventory, such as the computer model and manufacturer for Solaris x86).
dpkg-query	Linux	The following are searched in this order: • /bin • /usr/bin • /usr/local/bin.	*Command line:* `/successfulPath/dpkg-query -W --showformat=formatString` *Purpose:* Obtain a formatted list of packages identified in the dpkg database. ***Tip:*** *While the FlexNet inventory agent looks for this command on all Linux platforms (and runs it if present), it is typically only present on Debian/Ubuntu Linux distributions.* *Invoked using:* The account running the ndtrack executable (default: root).

Executable	Platform	Path	Notes
dspmq	All	Path(s) found in the process listing in which IBM MQ was identified.	*Command line:* `/successfulPath/dspmq -o all` *Purpose:* Reports as installation evidence the name (as ProductName) and active/inactive state (as EditionName, blank for active) of the queue managers on the system. Used by the Application Recognition Library to recognize IBM MQ (previously known as WebSphere MQ). *Invoked using:* An account determined by the following rules: • If the queue is active (so that the queue manager process is running), impersonate the user account that is running the queue manager process, and execute dspmq from the path used by the process. • When the queue is inactive, execute dspmq as the owner of that executable. The method of discovering the executable depends on the operating system configuration: ◦ If chown is enabled for non-root accounts (for example, on HP-UX), the dspmq path is identified in the /etc/opt/mqm/mqinst.ini configuration file. ◦ When chown is *not* enabled for non-root accounts, the dspmq path is identified by the first of the following methods to be successful: 1. Examining /opt/mqm 2. Looking in the /etc/opt/mqm/mqinst.ini file 3. Checking results of any file system scan (if run).
dspmqver	All	Path(s) found in the process listing in which IBM MQ was identified.	*Command line:* `/successfulPath/dspmqver` *Purpose:* Collect the IBM MQ (previously known as WebSphere MQ) version and build information for inclusion in inventory. *Invoked using:* An account determined by the same rules as described above for dspmq.

Executable	Platform	Path	Notes
`eeprom`	Solaris	`/usr/sbin`	*Command line:* `/usr/sbin/eeprom nvramrc` *Purpose:* Examines the contents of NVRAMRC to collect the chassis serial number. *Invoked using:* The account running the `ndtrack` executable (default: `root`). If you use a non-privileged account to run the tracker, the SPARC model information may be incorrect in inventory (for non-privileged accounts, the data collected is the value for `sysinfo SI_PLATFORM`, which is sometimes inconsistent).
`entstat`	AIX	`$PATH`	*Command line:* `/successfulPath/entstat adapter` *Purpose:* Reports the device type and MAC address of the network interface(s). *Invoked using:* The account running the `ndtrack` executable (default: `root`).
`getconf`	HP-UX	`/usr/bin`	*Command line:* `/usr/bin/getconf CPU_CHIP_TYPE` *Purpose:* Reports the type of the central processor in the server, for inclusion in hardware inventory. *Invoked using:* The account running the `ndtrack` executable (default: `root`).

`ifconfig`	All	`/usr/sbin` or `$PATH`	*Command lines:* On all platforms except HP-UX:
			`/successfulPath/ifconfig -a`
			On HP-UX:
			`/successfulPath/ifconfig adapter`
			Purpose: Lists all network interfaces; or reports the configuration of the interface identified as *`adapter`*.
			Invoked using: The account running the `ndtrack` executable (default: `root`). If running as an account other than root, information is less complete (for example, no MAC addresses for network adapters).

Executable	Platform	Path	Notes
ioscan	HP-UX	/usr/sbin	*Command line:* `/usr/sbin/ioscan -k -F -n` *Purpose:* Scans the kernel for data about installed hardware and I/O options, for inclusion in the hardware inventory data. *Invoked using:* The account running the ndtrack executable (default: root).
isainfo	Solaris	/usr/bin	*Command line:* `/usr/bin/isainfo -kv` *Purpose:* Determines the system architecture (32-bit or 64-bit) and related kernel information to include in inventory reporting. *Invoked using:* The account running the ndtrack executable (default: root).

java	All	Path(s) found in the file system scan in which java was identified.	*Command line:*
			`java -version`
			`java -fullversion`
			`java -XshowSettings -version`
			Purpose: Determines the Java product name, version information, and publisher.

kctune	HP-UX	/usr/sbin	*Command line:* `/usr/sbin/kctune lcpu_attr` *Purpose:* Reports whether hyperthreading is enabled on the system. *Invoked using:* The account running the ndtrack executable (default: root).
lanscan	HP-UX	/usr/sbin	*Command line:* `/usr/sbin/lanscan` *Purpose:* Collects the name and MAC address of each network adapter. Names are passed to ifconfig (see above). *Invoked using:* The account running the ndtrack executable (default: root).

Executable	Platform	Path	Notes
`lparstat`	Linux/ppc64le	`/usr/sbin`	*Command line:* `/usr/sbin/lparstat -i` *Purpose:* Used to query logical partition data on Linux Power machines that use logical partitions. *Invoked using:* The account running the `ndtrack` executable as `root` user.
`lppchk`	All	`/usr/bin`	*Command line:* `/usr/bin/lppchk -c packageName` *Purpose:* Performs a check of an installed AIX `lpp` package to ensure it is in a healthy state. Used to validate the package for the installed FlexNet inventory agent. *Invoked using:* The account running the `ndtrack` executable (default: `root`).
`lsbom`	OS X	`/usr/bin`	*Command line:* `/usr/bin/lsbom -p f path` *Purpose:* Obtains a listing of files identified within *path* by the installer's Bill of Materials (binary bom file). *Invoked using:* The account running the `ndtrack` executable (default: `root`).
`lscfg`	AIX	`$PATH`	*Command line:* `/successfulPath/lscfg -p` *Purpose:* Reports details about the video controller information (on AIX) for inclusion in hardware inventory. *Invoked using:* The account running the `ndtrack` executable (default: `root`).
`lscpu`	Linux/(ppc64le, s390x, aarch64/ arm64)	`/usr/bin`	*Command line:* `/usr/bin/lscpu` *Purpose:* Used to query CPU core data. *Invoked using:* The account running the `ndtrack` executable as `root` user.

Executable	Platform	Path	Notes
`lsnrctl`	All	`$ORACLE_HOME/bin`	*Command line:* `$ORACLE_HOME/bin/lsnrctl` *Purpose:* Invokes the Oracle Listener Control utility against a running listener to gather its network port address and the services (local and remote database instances) to which it provides access. *Invoked using:* Impersonation of the account running the `tnslsnr` service. (Impersonation requires that the `ndtrack` executable is running as `root`, without which Oracle discovery and inventory are disabled.)
`lspci`	Linux	`/sbin`	*Command line:* `/sbin/lspci` *Purpose:* Reports details about the video controller information (on Linux) for inclusion in hardware inventory. *Invoked using:* The account running the `ndtrack` executable (default: `root`).
`machinfo`	HP-UX	`/usr/contrib/bin`	*Command line:* `/usr/contrib/bin/machinfo` *Purpose:* Reports information about the machine processor. *Invoked using:* The account running the `ndtrack` executable (default: `root`).
`netstat`	All	`$PATH`	*Command line:* `/successfulPath/netstat -nr` *Purpose:* Collects the default IP gateway address. *Invoked using:* The account running the `ndtrack` executable (default: `root`).

Executable	Platform	Path	Notes
`osdbagrp`	All	`$ORACLE_HOME/bin`	*Command line:* `$ORACLE_HOME/bin/osdbagrp` *Purpose:* Identify the OS group for which each Oracle database instance has been configured. Used to provide logging information and allow warnings about potential issues running `sqlplus`. *Invoked using:* Impersonation of an account from the process list running either a database instance or a listener service from the same installation path as the `osdbagrp` executable being invoked.
`oslevel`	AIX	`$PATH`	*Command line:* `/successfulPath/oslevel -r` *Purpose:* Reports the operating system level, determined by examining a known set of Authorized Program Analysis Reports (APARs) supplied with the operating system. *Invoked using:* The account running the `ndtrack` executable (default: `root`).

`parstatus` and `vparstatus`	HP-UX	`/usr/sbin`	*Command line:*
			`/usr/sbin/parstatus -wM`
			`/usr/sbin/parstatus -CM`
			`/usr/sbin/vparstatus -wM`
			`/usr/sbin/vparstatus -M`
			`/usr/sbin/vparstatus -AM`
			Purpose: The `parstatus` command retrieves information about the nPartitions or hardware within a server, for inclusion in the hardware inventory data. The `vparstatus` version collects information about virtual partitions and their available resources (effectively, reporting on 'virtual machines').
			Invoked using: The account running the `ndtrack` executable (default: `root`). If a non-root account is used, `vparstatus` cannot be used, and inventory details including VMType, VMName and vPar capacity are lost.

Executable	Platform	Path	Notes
pkg	Solaris	/usr/bin	*Command line:* `/usr/bin/pkg contents -H -s pkg.fmri -o pkg.fmri,action.raw -tset -tfile -tlink -thardlink` *Purpose:* Identify the contents (including actions and attributes) of packages installed on the target device and registered in the Image Packaging System (IPS), specific to Solaris 11. This data is included in software inventory. *Invoked using:* The account running the ndtrack executable (default: root).
pkginfo	Solaris	$PATH	*Command line:* `/successfulPath/pkginfo -l name` *Purpose:* Gathers information about the named software package. *Invoked using:* The account running the ndtrack executable (default: root).

pkgutil	OS X	/usr/sbin	*Command line:* To collect details of a package:
			`/usr/sbin/pkgutil --pkg-info-plist packageName`
			To list the files for a package:
			`/usr/sbin/pkgutil --files packageName`
			Purpose: Collects details of packages and the files they contain to include in software inventory.
			Invoked using: The account running the ndtrack executable (default: root). If an account other than root is used, some OS X bundles under /Applications or /System/Library that are not accessible by the executing user cannot be reported in inventory.

Executable	Platform	Path	Notes
`ps`	AIX, Solaris	`/bin`	*Command line:* `/bin/ps -e -opid= -oruid= -ocomm=` *Purpose:* A fail-over step to identify processes that are required in later inventory gathering, when these could not be recovered from the `proc` file system. *Invoked using:* The account running the `ndtrack` executable (default: `root`).
`ps`	HP-UX	`/bin`	*Command line:* `/bin/ps -ef -opid= -oruid= -oargs=` *Further notes:* See initial entry for `ps` above. Note that the `ps` command is always required on HP-UX.
`ps`	Linux	`/bin`	*Command line:* `/bin/ps -e -opid= -oruid= -ocommand=` *Further notes:* See initial entry for `ps` above.
`ps`	OS X	`/bin`	*Command line:* `/bin/ps -ax -o pid,ruid,command` *Further notes:* See initial entry for `ps` above.
`rpm`	AIX, Linux	The following are searched in this order: • `/bin` • `/usr/bin` • `/usr/local/bin` • `/opt/freeware/bin` • `/opt/sfw/bin` • `/opt/local/bin`.	*Command line:* `/successfulPath/rpm --query --all --queryformat format` *Purpose:* Obtain a formatted list of packages from the Red Hat Package Manager. The multiple paths are mostly required for AIX. *Invoked using:* The account running the `ndtrack` executable (default: `root`).

Executable	Platform	Path	Notes
`setboot`	HP-UX	`/usr/sbin`	*Command line:* `/usr/sbin/setboot` *Purpose:* Reports whether hyperthreading is available on the system. ***Note:*** *For efficiency, `setboot` is only used when the `kctune` command returns a positive result. (This second call is not redundant on certain older versions of the OS.)* *Invoked using:* The account running the `ndtrack` executable (default: `root`).
`sh`	All	`/bin`	*Command line:* `/bin/sh -c script` *Purpose:* Runs the named script that has been delivered within `InventorySettings.xml` (these scripts may be updated through the Application Recognition Library). These scripts provide specialized inventory-gathering steps for use with Oracle products. They include the Oracle GLAS scripts required for preparing an Oracle audit report. *Invoked using:* The account running the `ndtrack` executable (default: `root`).

Executable	Platform	Path	Notes

sqlplus	All	$ORACLE_HOME/bin	*Command line:* Variations based on preference settings discussed below:
			`$ORACLE_HOME/bin/sqlplus "/ as sysdba"`
			`$ORACLE_HOME/bin/sqlplus "/ "`
			Purpose: Perform queries against running Oracle database instances to gather inventory on the Oracle Database product. (For ways that the tracker identifies $ORACLE_HOME, see the topic *How Agent-Based Collection of Oracle Inventory Works* in the *FlexNet Manager Suite System Reference* PDF.) This Oracle utility is invoked by a script delivered within InventorySettings.xml (described in the entry for sh).
			Invoked according to: The following rules:
			• If ndtrack is running as any account other than root, discovery of, and gathering inventory for, Oracle databases are both disabled on the target device.
			• When ndtrack is running as root, settings for the two preferences OracleInventoryAsSysdba and OracleInventoryUser determine the behavior, as follows (or for more detail, see OracleInventoryAsSysdba and OracleInventoryUser).
			1. If OracleInventoryAsSysdba=True (or omitted), the first command line shown above is used (with the parameter "/ as sysdba"). The account used depends on the value of the other preference:
			◦ If OracleInventoryUser is configured, the command is invoked impersonating that nominated account, with the database connection being made with the SYSDBA privilege (see OracleInventoryUser for requirements).
			◦ If OracleInventoryUser is not configured (the default), the command is invoked impersonating the account that is running the database instance, with the database connection being made with the SYSDBA

Executable	Platform	Path	Notes
			privilege. 2. If `OracleInventoryAsSysdba=False`, the second command line shown above is used (with the parameter `"/ "`). The accounts used depend on the value of the other preference: ◦ If `OracleInventoryUser` is configured, the command is invoked impersonating that nominated account, with the database connection being made as the same `OracleInventoryUser` account (see OracleInventoryUser for requirements). ◦ If `OracleInventoryUser` is not configured, Oracle inventory collection is not supported. ***Note:*** *This approach means that the tracker can collect inventory only from running database instances. Instances that are discovered, but are not running at inventory time, are reported in the task status: navigate to the discovered device properties, select the* ***Status*** *tab, and expand the* ***Oracle database inventory*** *heading.*
`swlist`	HP-UX	`/usr/sbin`	*Command line:* `/usr/sbin/swlist -v -lproduct -atag -arevision -atitle -ainstall_date -avendor_tag -asize -aarchitecture -ais_patch -lfile -apath -atype` *Purpose:* Obtains a listing of software products installed on the local host. *Invoked using:* The account running the `ndtrack` executable (default: `root`). If, instead, you choose to run the tracker using a non-privileged account, installation evidence will be missed where access has been restricted.

Executable	Platform	Path	Notes
`vxlicrep`	All	`/sbin`	*Command line:* `/sbin/vxlicrep` *Purpose:* Creates installation evidence used by the Application Recognition Library to recognize installations of Symantec. *Invoked using:* The account running the `ndtrack` executable (default: `root`).

`xl`	Linux	The following are searched in this order:
		• `/sbin`
		• `/usr/sbin`
		• `/usr/local/sbin`
		• `/bin`
		• `/usr/bin`
		• `/usr/local/bin`.
			Command lines:
			`/successfulPath/xl info -n`
			`/successfulPath/xl vm-list`
			`/successfulPath/xl list-vm`
			Purpose: This Xen management tool reports any guest domains (virtual machines) present on the server. This information assists in correctly reporting device inventory, including the mapping between host devices and virtual devices.
			Invoked using: The account running the `ndtrack` executable (default: `root`).

`zoneadm`	Solaris	`/usr/sbin/`	*Command line:* `/usr/sbin/zoneadm list -p` *Purpose:* Provides the list of zones that are running inside the global zone (and therefore is run only inside the global zone). Inventory includes the name and UUID of each zone. *Invoked using:* The account running the `ndtrack` executable (default: `root`).

Executable	Platform	Path	Notes
zonecfg	Solaris	/usr/sbin/	*Command line:* `/usr/sbin/zonecfg -z {zonename} info {dedicated-cpu|capped-cpu|pool}` *Purpose:* Provides configuration information about the specified zone, and specifically its resource management method (`dedicated-cpu`, `capped-cpu`, or resource `pool`). This command is run only inside the global zone. *Invoked using:* The account running the `ndtrack` executable (default: `root`).

Common: Child Processes on Windows Platforms

In all of the Adopted case, the Agent third-party deployment case, and the Zero-footprint case, the tracker always runs as `LocalSystem`, because elevated privileges are required to complete several aspects of inventory gathering. In the Core deployment case or the FlexNet Inventory Scanner case, it is possible to run the tracker under a different account, but best practice is to run it with administrator privileges, or you may lose inventory functionality.

Note: *On Microsoft Windows, the tracker does not prevent invocation by an account that has lesser privileges; but you would then need to ensure that such an account had all the required access rights for the kinds of inventory you expected to gather on a target device. Since this is highly dependent on your environment, this approach is unsupported.*

Since the tracker always runs with elevated privileges, it is important that it only acts in place of accounts that are known and trusted in your environment. In many cases, the commands or services are already running as `LocalSystem` on your Oracle server(s), so there is no effective change when the tracker does the same. But with Oracle Database 12c, or with IBM MQ (previously WebSphere MQ), it is possible that a service account has been used. To ensure that only actions by accounts that are trusted are also run by the tracker, it relies on details found in the Windows registry and in Windows Service Control Manager (SCM), both of which can only be modified by a system administrator.

In summary:

- Commands in safe system paths (not writable by other users) are run as `LocalSystem`.
- Commands found within paths listed in the `%PATH%` environment variable for the `LocalSystem` user are run as `LocalSystem`.

  Note: *This makes it important that, as is normal secure practice, you do not allow any unsecured directories to be included in the `%PATH%` environment variable for the `LocalSystem` user.*
- Other necessary commands and utilities are run as `LocalSystem` only if:

- They are normally executed by accounts trusted in your Windows SCM configuration, or
- They are saved in paths recorded in Oracle keys or IBM MQ keys in the Windows registry.
- Specifically for `java.exe` commands, the PerformOracleJavaAuditScan preference is enabled and the `java.exe` being considered is digitally signed.

The table of child processes on Windows is organized in alphabetical order of the executables invoked by the tracker.

Tip: *All child processes are invoked in hidden mode.*

Executable	Path	Notes
`cmd`	`C:\Windows\System32`	*Command line:* `C:\Windows\System32\cmd.exe script` *Purpose:* Runs the named script that has been delivered within `InventorySettings.xml` (these scripts may be updated through the Application Recognition Library). These scripts provide specialized inventory-gathering steps for use with Oracle products. They include the Oracle GLAS scripts required for preparing an Oracle audit report. *Invoked using:* The account running the `ndtrack` executable (default: `LocalSystem`).
`db2licm.exe`	Path(s) for IBM Db2 found in the Windows registry `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\DB2\InstalledCopies`	*Command line:* `\successfulPath\bin\db2licm.exe -l / -g` *Purpose:* Reports inventory of the IBM Db2 Database (including its product identifier) and its optional add-ons, including the available license information. *Invoked using:* The account running the `ndtrack` executable (normally `LocalSystem`). With the parameters shown above, IBM Db2 on Microsoft Windows allows this command without an elevated account; although with certain additional parameters, an elevated account becomes necessary.
`db2ilist.exe`	Path(s) found in the Windows registry for IBM Db2.	*Command line:* `\successfulPath\bin\db2ilist.exe` *Purpose:* Lists all the database instances running in the context where `db2ilist` is executed (normally, instances from the same database installation that provides the `db2ilist`executable). *Invoked using:* The account running the `ndtrack` executable (normally `LocalSystem`, although IBM Db2 does not require elevated privileges for this command on Windows).

Executable	Path	Notes
`dspmq`	Path(s) found in the Windows registry for IBM MQ.	*Command line:* `\successfulPath\dspmq -o all` *Purpose:* Reports as installation evidence the name (as `ProductName`) and active/inactive state (as `EditionName`, blank for active) of the queue managers on the system. Used by the Application Recognition Library to recognize IBM MQ (previously known as WebSphere MQ Manager). *Invoked using:* The account running the `ndtrack` executable (default: `LocalSystem`).
`dspmqver`	Path(s) found in the Windows registry for IBM MQ.	*Command line:* `\successfulPath\dspmqver` *Purpose:* Collect the IBM (or WebSphere) MQ version and build information for inclusion in inventory. *Invoked using:* The account running the `ndtrack` executable (default: `LocalSystem`).

`java`	Path(s) found in the file system scan in which java was identified.	*Command line:*
		`java -version`
		`java -fullversion`
		`java -XshowSettings -version`
		Purpose: Determines the Java product name, version information, and publisher.
		Further notes: Only executed if the PerformOracleJavaAuditScan preference is enabled and the `java.exe` being considered is digitally signed.

`lsnrctl`	`%ORACLE_HOME%\bin`	*Command line:* `%ORACLE_HOME%\bin\lsnrctl` *Purpose:* Invokes the Oracle Listener Control utility against a running listener to gather its network port address and the services (local and remote database instances) to which it provides access. *Invoked using:* The account running the `ndtrack` executable (default: `LocalSystem`).

Executable	Path	Notes
nbtstat	%PATH%	*Command line:* `\%PATH%\nbtstat -A IPAddr` *Purpose:* Returns the local NetBIOS name table for the computer at the nominated IP address, as well as the MAC address of the adapter card connecting it to the network. This data is used in discovery. *Invoked using:* The account running the `ndtrack` executable (default: `LocalSystem`).
powershell	On 64-bit systems: `%SystemRoot%\system32\WindowsPowerShell\v1.0`and on 32-bit systems: `%SystemRoot%\SysWOW64\WindowsPowerShell\v1.0`	*Command line:* `\platformPath\powershell.exe` *Purpose:* Runs the named script that has been delivered within `InventorySettings.xml` (these scripts may be updated through the Application Recognition Library). These scripts provide specialized inventory-gathering steps for use with Oracle products. They include the Oracle GLAS scripts required for preparing an Oracle audit report. *Invoked using:* The account running the `ndtrack` executable (default: `LocalSystem`).

Executable	Path	Notes
sqlplus	%ORACLE_HOME%\bin	*Command line:* `%ORACLE_HOME%\bin\sqlplus "/ as sysdba"` *Purpose:* Perform queries against running Oracle database instances to gather inventory on the Oracle Database product. (For ways that the tracker identifies %ORACLE_HOME%, see the topic *How Agent-Based Collection of Oracle Inventory Works* in the *FlexNet Manager Suite System Reference* PDF.) This Oracle utility is invoked by a script delivered within InventorySettings.xml (described in the entry for cmd). *Invoked using:* The account running the ndtrack.exe executable (default: LocalSystem). The account running ndtrack must be a member of the ora_dba security group for the target Oracle Database (where the LocalSystem account is displayed as NT_AUTHORITY\SYSTEM; and if this account is missing, it must be entered as SYSTEM). ***Tip:*** *From Oracle Database 12c, there is a distinct ora_dba group for each separate %ORACLE_HOME%.* ***Note:*** *This approach means that the tracker can collect inventory only from running database instances. Instances that are discovered, but are not running at inventory time, are reported in the task status: navigate to the discovered device properties, select the* ***Status*** *tab, and expand the* ***Oracle database inventory*** *heading.*
vxlicrep	File path extracted from %VCS_ROOT%.	*Command line:* `\successfulPath\VRTSsfmh\bin\vxlicrep.exe` *Purpose:* Creates installation evidence used by the Application Recognition Library to recognize installations of Symantec. *Invoked using:* The account running the ndtrack executable (default: LocalSystem).

Common: Supporting Mutual TLS

Communication using the HTTPS protocol between a client (such as a target inventory device) and a server (such as an inventory beacon) is secured by Transport Layer Security (TLS). Traditionally this has been standard (or single-sided) TLS, where the server has a certificate that the client must verify before communicating. In more security-conscious environments, it may be necessary not only to validate that we are communicating with the correct server, as proven by

its valid certificate, but also to be certain that only an authorized client device can join the communication. Enter *mutual* TLS, where both the client and the server must be authorized by separate valid certificates before the communication may proceed.

Setting up the inventory beacon for the server side of mutual TLS is documented in the online help (see ***FlexNet Manager Suite Help > Inventory Beacons > Local Web Server Page > Configuring Mutual TLS***.) In contrast, this topic provides a few introductory notes for setting up the client side of mutual TLS on your target inventory device. In fact, there are many possible methods for obtaining and managing client-side certificates, and your enterprise may already have its preferred process. In that case, use your preferred process. FlexNet Manager Suite does not mandate any particular process, nor does it provide tools for managing, distributing, or installing client-side certificates.

Remember: *Once an inventory beacon has been configured for mutual TLS (specifically, configured to require a client-side certificate before communicating), it is impossible for an inventory device that does not have a client-side certificate to communicate with that inventory beacon for any reason:*

- *It cannot download device policy, schedule changes, or software updates*
- *It cannot upload any status changes, nor any collected discovery results or inventory files.*

Also keeping in mind that the locally-installed FlexNet inventory agent is not tied to a particular inventory beacon, but assesses for each download/upload which inventory beacon is the most appropriate (for example, has the fastest response times), the decision to implement mutual TLS is typically a system-wide one (or at least, one that covers all devices within distinct boundaries, such as clearly defined subnets).

You only need one client-side certificate, as the same certificate (after export to the appropriate format) can be distributed to multiple inventory devices. Unlike the case with server-side certificates, you do not need a separate root certificate that attests to the Certificate Authority itself for this client-side certificate.

Setting up client-side certificates for mutual TLS (overview):

1. Obtain your client-side certificate from an appropriate Certificate Authority.

   One way to do this is to prepare a certificate signing request (CSR) in the same way (perhaps even at the same time) as you prepare one for a server-side certificate for one of your inventory beacons, using IIS on that server (for details, see ***Configuring Mutual TLS*** in the online help, step 7).

   An alternative method is to use the MMC snap-in for Certificates, which has the advantage of running on any Windows computer where you have administrator rights. Instructions can be found online (for example, https://www.msb365.blog/?p=3886).

2. When the certificate is received back from the Certificate Authority, import it into the **Certificates (Local Computer) > Personal** store on the same working computer (perhaps your inventory beacon) where you created the CSR.

   In this case, this certificate is *not* for attaching to the IIS bindings on the inventory beacon, and we import it into a *personal* store just to facilitate exporting in the appropriate formats for your Windows and UNIX-like inventory devices.

3. From the working store, export the client-side certificate (including the private key) into `.pfx` format. For example:

   a. Right-click the certificate file in the list for your chosen store, and select **All Tasks > Export...** to start the **Certificate Export Wizard**.

b. On the welcome page, click **Next**.

c. On the **Export Private Key** page, select **Yes, export the private key**, and click **Next**.

d. On the **Export File Format** page, select **Personal Information Exchange – PKCS #12 (.PFX)** and then check **Include all certificates in the certification path if possible**. Click **Next**.

e. On the **Security** page, select **Password**, and create and confirm the password (keep a note of this password in your enterprise-approved system, as it is required when you import the certificate with private key to other Windows devices).

f. Click **Next**, and in the **File to Export** page, browse to a path and add a file name to save your exported certificate. Click **Next**.

g. On the final page of the wizard, confirm your settings, and click **Finish**.

4. Deploy the exported client-side `.pfx` file to target inventory devices running a Windows operating system, using your preferred method (and having the password you created to protect the private key available for the installation process).

Tip: *For the server-side certificate, recall that the root certificate for your CA must be in the* ***Trusted Root Certification Authorities*** *store on both the inventory beacon and all the client inventory devices that are communicating with the inventory beacon. This may mean that you have an opportunity to deploy both certificates (the root CA certificate, and your client-side certificate) to your Windows-based inventory devices within the same process.*

5. For target inventory devices running UNIX-like operating systems, a different format is required that provides the certificate and the private key in separate `.pem` files. One method of conversion is to use the `openssl` toolkit, available through https://www.openssl.org/source/, on a convenient Windows device where you have `openssl` and a copy of the `.pfx` file you are deploying for Windows devices.

a. To extract the private key from the `.pfx` file to a `.pem` file (this file still includes the password protection):

```
openssl pkcs12 -in filename.pfx -nocerts -out key_with_pass.pem
```

b. To remove the password from the private key file just created:

```
openssl rsa -in key_with_pass.pem -out client_key.pem
```

c. To export the certificate without the private key (but still including the necessary public key) in a `.pem`file:

```
openssl pkcs12 -in filename.pfx -clcerts -nokeys -out client_cert.pem
```

d. Open the resulting certificate file in your preferred flat text editor (such as Notepad), delete all preliminary lines of text before `-----BEGIN CERTIFICATE-----`, and save the amended file.

6. Use your preferred deployment method to install both files on target inventory devices running UNIX-like operating systems:

- Deploy the certificate file to `/var/opt/managesoft/etc/ssl/client/client_cert.pem`
- Deploy the key file to `/var/opt/managesoft/etc/ssl/client/private/client_key.pem`.

Tip: *If you are deploying new devices where you are installing FlexNet inventory agent for the first time, you can include both the certificate file and the key file in the deployment package, as described in Agent Third-Party Deployment: Installing FlexNet inventory agent on UNIX.*

7. For target inventory devices running UNIX-like operating systems, you must also set three preferences to their new values in the `config.ini` file that serves as a pseudo-registry on these devices. For more details, see:
   - AddClientCertificateAndKey (which must be set to `True`)
   - SSLClientCertificateFile (which must give the file path for the `.pem` certificate file)
   - SSLClientPrivateKeyFile (which must give the file path for the private key).

When the client-side certificates are deployed to your target inventory devices (along with the CA root certificates needed for validation of the server-side certificates), and the additional preferences have been set for UNIX-like platforms, you are ready to run this environment with mutual TLS verification of secure communications using the HTTPS protocol.

Common: Collection from Virtual Environments

FlexNet Manager Suite can collect inventory from a number of virtual and partitioned environments (collectively: virtualization), including the following virtual machine or partition types:

- Microsoft Hyper-V
- Linux KVM
- LPAR (logical partition) on IBM AIX
- nPAR (Hewlett-Packard hard partition with separate hardware facilities, each of which may also host vPARs)
- Oracle VM
- VMware
- vPar (Hewlett-Packard software partition, a virtual machine)
- Zone (non-global zones on Solaris 10 and later).

(In addition, virtualization inventory can be collected by separate adapters for Citrix Virtual Apps and Virtual Desktops (previously XenApp and XenDesktop), but this topic is focused only on inventory collection that may involve the `ndtrack` executable.)

In each case, the business goal is to review relationships that show which virtual machines are operating on which hosts (as well as the application software in use). Obviously, there are different techniques across these different technologies, falling into two main classes:

- For hypervisor virtualization, inventory is collected from the host (or hypervisor), and also collected separately from each of the virtual machines (which, in most systems, are ignorant about their host machine). Then common data from the BIOS for the guest OS (that is, the serial number of the virtualized hardware) is matched to the host's list of guest serial numbers, allowing FlexNet Manager Suite to present a structured set of records of virtual machines

related to their hosts. This is the approach used for Hyper-V, Linux KVM, Oracle VM, and VMware (although for Linux KVM, the guest's UUIDs are used as serial numbers). In the case of VMware, the VMware host inventory is collected remotely by an inventory beacon (using the API available on either a vCenter management server or an ESX host). In other cases, the host inventory may be gathered either by an installed FlexNet inventory agent, or by zero footprint inventory collection by an inventory beacon.

Tip: *For Linux KVM, by default the host inventory includes identification of its guest VMs. When, as usual, the installed FlexNet inventory agent is running as* `root` *on the host, the uploaded host inventory identifies all guest VMs on the host. If, for some reason, you are executing FlexNet inventory agent under a different user, the uploaded inventory only identifies those guest VMs for which that same user account has both access rights and viewing rights through the* `libvirt` *API. You can confirm those viewing rights by executing the following command as the non-*`root` *user account:*

```
$ virsh list --all
```

- For container virtualization, no inventory is required (or taken) for the virtualization host. Inventory is collected from each of the partitions/zones/containers, which includes the resource allocations for that partition, its unique identifiers, and information about the physical host that the partition is running on (including the host's serial number). When the inventories have been imported, FlexNet Manager Suite compares common values in the records to fabricate a host record (since no direct inventory has been collected from the host itself). Because the partitioned host does not supply a machine name, the fabricated record uses the serial number (again) as the machine name. (For Solaris, the global zone is inventoried particularly for the processor, core and thread counts of the host server; and a VM host record is then fabricated to include both global and non-global zones on the host server.)

These differences have the following general implications for deployment of `ndtrack`. These implications are common for both the full FlexNet inventory agent and the FlexNet inventory core components, and regardless of deployment method:

- For hypervisor virtualization like Hyper-V, Linux KVM, and OracleVM, `ndtrack` must be deployed to every virtual machine *and* to the host server (or, as shown in the table below, zero footprint inventory collection may be a useful alternative to the latter). In the absence of inventory imported from the host server (or if you subsequently delete the host device record), the virtual machine records are left unlinked, without a host.

Tip: *In the case of Linux KVM, if you disable the default collection of virtualization data from the host even when other inventory may still be collected from the same server (using the PerformKvmInventory preference), the guest VMs cannot be linked to their host; and the host cannot be identified as a* `VMHost`*, and instead its* ***Inventory device type*** *is shown as* `Computer`*.*

- For most container virtualization, as on AIX using LPARs, `ndtrack` must be deployed to every partition, and should *not* be deployed to the host.

However, there are specific exceptions to this general division by virtualization type, so the following listing makes specific whether to collect inventory from the host, and if so by what methods (using the short-hand labels defined in Deployment Overview: Where to, and How).

Technology	Host inventory required	Method(s)
Microsoft Hyper-V	Yes	Adopted, Agent third-party deployment, Zero-footprint.
Linux KVM	Yes	Adopted, Agent third-party deployment, Zero-footprint. ***Tip:*** *Linux KVM inventory is supported in version 14.0.0 or later of FlexNet inventory agent. For zero footprint inventory collection by an inventory beacon, version 14.0.0 or later of FlexNet Beacon is required. (Both these versions were released as part of FlexNet Manager Suite release 2019 R2.)*
LPAR (logical partition) on IBM AIX	No	n.a.
nPAR (HP-UX)	No	n.a.
Oracle VM	Yes	Adopted, Agent third-party deployment, Zero-footprint. Additional inventory is remotely collected (agentless) by an inventory beacon using an Oracle API. See note below.
VMware	Yes	Remote agentless inventory by an inventory beacon using the vCenter or ESX API.
vPar (HP-UX)	No	n.a.
Zone (Solaris 10 and later)	Yes (global zone)	Adopted, Agent third-party deployment, Zero-footprint.

Note: *Collecting complete inventory from Oracle VM requires two forms of host inventory collection:*

- *Agentless inventory remotely collected from the Oracle VM Manager by an inventory beacon identifies all managed VM hosts, their guest VMs, and any related cluster information. Clustering information is required for full capacity licensing of Oracle Database. However, Oracle VM Manager does not identify CPU affinity for any of the guest VMs.*
- *Inventory collected on the Oracle VM hosts by* `ndtrack` *(in either the Adopted case or Zero-footprint case) identifies the VM host, its guest VMs, and the relevant CPU affinity applicable to any of the VMs. Affinity information is required for subcapacity licensing of Oracle Database using Oracle Processor or Oracle NUP license types (and this has the potential for considerable cost savings over full capacity licensing). However, the individual Oracle VM hosts do not reveal details about clustering, needed for full capacity licensing.*

For these reasons, both forms of host inventory are available for an Oracle VM solution, so that both clustering and CPU affinity information is available. (If you use only full capacity licensing for Oracle Database, which may require you to license entire clusters, you could omit the `ndtrack` *inventory collection from each Oracle VM host.)*

Inventory for guest devices

For each of the technologies listed in the previous table, inventory from the guest virtual machine (or partition) can be collected by the `ndtrack` executable. This can be achieved through any of the established use cases:

- Installation of the FlexNet inventory agent on the virtual device in the Adopted case
- Installation of the FlexNet inventory agent on the virtual device in the third-party Agent third-party deployment case
- Zero-footprint inventory collection by the inventory beacon (where virtual machine up-time and network access make this practical)
- Your own deployment of the FlexNet inventory core components, either to the target device or to a network share, in the Core deployment case.

Common: Gathering Inventory from Solaris Zones

Calculating the license compliance position for Oracle Processor and IBM PVU licenses becomes complex when the applications are installed on Solaris zones as the license consumption depends on the number of processor threads assigned to a Solaris zone. For example, the license consumption for an Oracle Database Processor license depends on the maximum number of processor threads that can be used by the Solaris zone where the application has been installed.

To calculate the license consumption for these licenses, the Flexera inventory collection component (`ndtrack`, either deployed locally or run remotely) collects the hardware-specific information like the maximum number of processor threads and some other properties. This information is used to calculate the license position for some Oracle Processor, Oracle Named User Plus, and IBM PVU licenses.

Note: *To get the accurate consumption of IBM PVU, Oracle Processor, and Oracle Named User Plus licenses on Solaris zones, upgrade the following inventory gathering components to the 2017 R1 or later version:*

- *For adopted devices: FlexNet inventory agent(s)*
- *For zero footprint inventory collection: FlexNet Beacon(s).*

Compatibility matrix

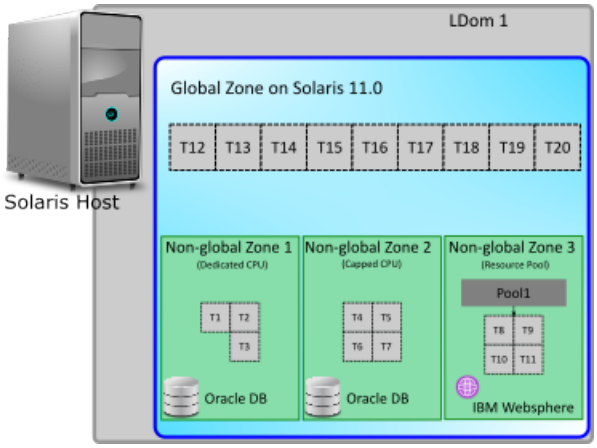
Solaris zones with the following environments are supported for inventory collection:

Zone property	Support
Architecture	• SPARC, including Logical Domains • X86

Zone property	Support
Resource management method for zones	• Resource pool • Capped CPU • Dedicated CPU

Common: Targeting for Solaris Zones

To calculate the compliance position for Oracle Processor and IBM PVU licenses, FlexNet Manager Suite needs hardware-specific information about the Solaris zone where the application has been installed. As the global zone manages and keeps track of the host resources used by the non-global zones, you must collect inventory from the global zone when you collect inventory from one or more non-global zones. Consider the following example architecture of a Solaris host:



The diagram illustrates a Solaris host with a logical domain (LDom 1) that contains a global zone and three non-global zones. Non-global Zone 1 uses three processor threads in a dedicated mode from the Logical Domain, Non-global Zone 2 uses four processor threads through the capped CPU resource management method, and Non-global zone 3 uses another four processor threads through the resource pool resource management method.

To calculate the compliance position for the Oracle Processor and IBM PVU licenses for the applications installed on Non-global Zone1, you need to inventory the Global Zone and Non-global Zone 1 through one or more discovery and inventory rules. The hardware details would be collected from the global zone, and the application inventory would be collected from the non-global zone. The same targeting technique applies for Non-global Zone 2 and Non-global zone 3. For more information on creating targets, see the online help.

Representation of Solaris architecture

The following table describes how different entities of Solaris architecture are displayed on the **Virtual Devices and Clusters** page :

Solaris entity	Representation
LDom	VM host
Processor set and Pool	Resource Pool

Solaris entity	Representation
Global or non-global zone	Virtual Machine

Note: *When a physical machine has been hard-partitioned into two LDoms, two default global zones are created. FlexNet Manager Suite collects an* `.ndi` *file for each global zone.*

Data sources for Solaris zones inventory

To get an accurate license compliance position for Oracle Processor and IBM PVU licenses for the applications installed on Solaris zones, Flexera recommends using its inventory collection component. A locally-installed FlexNet inventory agent is the most complete and capable format for use of this component.

License consumption calculations

An accurate license consumption for license metrics relying on hardware properties can only be calculated when the required data is available to FlexNet Manager Suite. For example, the license consumption calculation for an Oracle Processor or IBM PVU license requires processor, core, threads or similar from the host, pool, or the virtual machine itself. The virtualization hierarchy and properties (such as capped, uncapped, dedicated partitioning) are also required. Missing hardware details for any level of the virtualization hierarchy may lead to the following values for consumption on a Solaris zone:

- **Approximated value**: When the **Threads (max)** property is missing for a pool, FlexNet Manager Suite uses the capping value defined at the processor set level. If this value is also missing, the value at the host level is used.
- **Zero**: In all other cases.

The following table summarizes the impact of missing hardware details on Oracle Processor and IBM PVU license consumption calculations:

License type	Level	Missing property value	UI location	Consumption value
Oracle Processor	Host	Cores	The **Hardware** tab of the host properties.	0
	Host	Threads	The **Hardware** tab of the host properties.	0
	VM	Threads (max)	The **VM Properties** tab, under the **Virtual machine properties** section.	0
	Pool	Threads (max)	The **VM Properties** tab, under the **Pool properties** section.	Approximated value
IBM PVU	Host	Cores	The **Hardware** tab of the host properties.	0
	Host	Threads	The **Hardware** tab of the host properties.	0
	VM	Threads	The **Hardware** tab of the VM properties.	0

License type	Level	Missing property value	UI location	Consumption value
	Pool	Threads	The **VM Properties** tab, under the **Pool properties** section.	Approximated value

Inventory beacon settings for Solaris zones inventory

To collect inventory from Solaris zones through zero footprint inventory collection, the inventory beacon needs the credentials of the root user for these zones. For this, the Password Manager on the appropriate inventory beacon(s) needs to be updated with these credentials. Use the value `SSH account (password)` for the **Account Type** field while adding the root user to the Password Manager. For more information on using the Password Manager, see the online help for the inventory beacon.

Common: License Reconciliation Considerations for Processor-Based Licenses

The inventory data collected by the inventory component contains the details of the maximum and active processor threads used by each Solaris zone. If a resource pool has been configured, the maximum number of processor threads and the active number of processor threads are considered at the pool level. If no pool is configured, these properties are considered at the zone level. Wherever necessary in the virtualization hierarchy (host, pset, pool, and zone), the required resource capping is applied. For example, if a host has only 16 threads and the pool consumption sums up to 24 threads, only 16 would be considered as consumption.

To calculate the core count of a zone (when needed to cap the license consumption for Oracle and IBM licenses), the starting point is the maximum number of threads used in any non-global zone:

1. The `MaximumNumberOfLogicalThreads` per non-global zone (say TZ) is obtained from the non-global zone inventory.
2. The `NumberOfThreadsPerCore` for the host (say TC) is obtained from the global zone inventory.
3. The first number is divided by the second (TZ/TC) to get the number of cores for the zone.

Common: Importing Registry Settings

This topic applies to all forms of gathering FlexNet inventory, by any of the cases covered in this reference, for inventory devices running Windows only (does not apply to UNIX-like devices).

You may configure the FlexNet inventory agent to collect information from the registry on Windows devices, and include the results in the normal `.ndi` inventory file, uploaded through the inventory beacon hierarchy to your central inventory server (or, in smaller implementations, your application server). However, because imports from the registry are not a normal part of inventory collection, these uploads are not imported into the inventory database until you also customize the settings on your inventory server. Once the settings are customized to allow for import into the inventory database, the normal processes take over: the nightly full import transfers the data from the inventory database into the compliance database, and after the license compliance calculations, the results are available in the web interface of FlexNet Manager Suite, specifically in the inventory device property sheet **Evidence** tab, when you select the **WMI** sub-tab.

On the inventory device

On the inventory device, the FlexNet inventory agent uses the `IncludeRegistryKey` preference to control which areas of the registry it collects as part of inventory. You can set this preference in either of two ways:

- By including it on the command line for FlexNet inventory agent
- By configuring the registry entry `[Registry]\ManageSoft\Tracker\CurrentVersion\IncludeRegistryKey` on each inventory device where you want to collect registry values.

The default value of `IncludeRegistryKey` is configured to collect only information from the uninstall section of the registry (Add/Remove entries). This behavior is equivalent to configuring `IncludeRegistryKey` with the value:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
```

Important: *For correct collection of software inventory, the uninstall section (as above) must always be included in this preference. If you add more paths, separate them with a semicolon (;) or comma (,) between each path.*

Enter the keys you want to track in `IncludeRegistryKey`, separating paths with semicolons or commas. For example, to track settings related to the installation of Adobe Photoshop 7.0 (while maintaining collection of the uninstall section of the registry), set `IncludeRegistryKey` to:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Adobe\Photoshop\7.0\ApplicationPath;
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
```

Warning: *Collecting large amounts of registry information may significantly degrade the performance of inventory resolution on the central application server. For this reason, the amount of registry information you collect should be minimized and restricted to values that you really do require.*

For more information about this preference on inventory devices, see IncludeRegistryKey.

Configuring your central server

These changes must be made on your inventory server (or, in smaller implementations, whichever server is hosting this functionality, such as your application server).

There are two distinct ways that you can configure your inventory server for the import of registry settings from uploaded inventory:

- You can make a precision setting that exactly specifies one registry entry for collection (and you can repeat this process for as many registry settings as you want to track), as described in Targeting Individual Entries
- You can make a general setting that 'scoops up' unnamed registry entries and includes them in the import, for which see Collecting All Uploaded Entries.

FlexNet Manager Suite (On-Premises)

2023 R2

Targeting Individual Entries

Use this process (as many times as required) on your inventory server to specify an individual registry entry, collected

from an inventory device and uploaded as part of the inventory (`.ndi`) file, that you want imported into the inventory database. Thereafter, the next full inventory import and license compliance calculation (typically scheduled overnight) makes the specified registry setting visible in the **WMI** sub-tab of the **Evidence** tab of the inventory device properties.

To specify a registry entry for import:

1. On the inventory server, start `regedit`.
2. Navigate to *exactly* this path:

   ```
   HKLM\SOFTWARE\ManageSoft Corp\Managesoft\Reporter\Current Version\Registry
   ```

3. Create a sub-key with a meaningful name that allows for easy maintenance.

   For example, you might name this key after the related software product that is being tracked. If you are tracking the Flexera uploader on an inventory device, you might name this tracking key something like `IDUploader`.

4. Under this key, insert the following values (name/data pairs):

Name	Data
ClassType	Mandatory (if you do not correctly create this value, the import of an individual registry entry fails). The **ClassType** must be set to exactly this value: `Win32_Registry`
InterestedPath	Specify the path of the registry key that is to be imported from inventory. ***Tip:*** *You may use either the full form of the key root (such as `HKEY_LOCAL_MACHINE`) or its short form (such as `HKLM`).*
InterestedName	Name the entry within the specified key that is to be imported.
Description	This will be displayed in the web interface as both the **Property name** and **Raw property name** values. You must specify this special value: `$Path` This uses the full path of the registry entry as its display name. ***Tip:*** *Other values that can be used in different settings may not be used when targeting an individual registry entry.*

5. Save your changes, and close `regedit`.

> ***Example:*** To display the location set for the log files for the uploader component of FlexNet inventory agent (on the inventory device, this is recorded in *`[Registry]`*`\ManageSoft\ Uploader\CurrentVersion\`

LogFile), you might configure the following registry values under *[Registry]*\ManageSoft\Reporter\CurrentVersion\Registry\IDUploader on your inventory server:

Name	Data
ClassType	Win32_Registry
InterestedPath	HKLM\SOFTWARE\ManageSoft Corp\ManageSoft\Uploader\CurrentVersion
InterestedName	LogFile
Description	$Path

After the next full inventory import and license compliance calculations, this is displayed in the **WMI** sub-tab of the **Evidence** tab of the inventory device properties, and likely displays the default value of $(TempDirectory)\ManageSoft\uploader.log.

FlexNet Manager Suite (On-Premises)

2023 R2

Collecting All Uploaded Entries

Use this process on your inventory server to specify import of all remaining registry entries collected from an inventory device and uploaded as part of the inventory (.ndi) file. Thereafter, the next full inventory import and license compliance calculation (typically scheduled overnight) makes the registry settings visible in the **WMI** sub-tab of the **Evidence** tab of the inventory device properties.

Caution: *Collecting large amounts of registry information may significantly degrade the performance of inventory resolution on the central application server. For this reason, the amount of registry information you collect should be minimized and restricted to values that you really do require. For information on targeting individual registry entries (rather than scooping up all uploaded registry entries), see Targeting Individual Entries.*

To specify import of all uploaded registry entries:

1. On the inventory server, start regedit.
2. Navigate to *exactly* this path:

   ```
   HKLM\SOFTWARE\ManageSoft Corp\Managesoft\Reporter\Current Version\Registry
   ```

3. Under this key, insert the following values (name/data pairs):

Name	Data
SaveRegistryClass	Either omit this setting, or set **SaveRegistryClass** appropriately for the **ClassType** to be imported and displayed with the WMI data. As this imported value must be set to exactly: `Win32_Registry` it may be safer to omit the setting, in which case the FlexNet inventory agent sets this as the default value, and uploads it appropriately, ready for import.
SaveRegistry	`True` This tells FlexNet Manager Suite to import all registry entries that have not already been used for software inventory (the uninstall details) or specified for import individually (see Targeting Individual Entries).

4. If you want to exclude certain registry keys from being imported, list the exclusions in the following registry key:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\ManageSoft
      Corp\ManageSoft\Reporter\CurrentVersion\SaveRegistryExclude
   ```

   By default, **SaveRegistryExclude** is set to `HKLM\SOFTWARE\ Microsoft\Windows\CurrentVersion\ Uninstall`. This value excludes Add/Remove Programs entries from the registry keys imported in WMI entry. (These Add/Remove Programs entries are defined by default in the **IncludeRegistryKey** settings on inventory devices for import as software inventory records.)

   Tip: *Notice that:*

   - *Values for any registry keys may optionally use the HKLM short form for the key root value*
   - *If you add to the **SaveRegistryExclude** list, be sure to keep the default value present as well so that you do not include software inventory in WMI reporting*
   - *Use a semi-colon (;) or comma (,) to separate each key if you wish to list multiple registry keys for exclusion.*

5. Save your changes, and close `regedit`.

The next full import (typically nightly, associated with the license consumption calculations) makes all imported registry settings visible in the **WMI** sub-tab of the **Evidence** tab of the inventory device properties.

FlexNet Manager Suite (On-Premises)

2023 R2

Common: Acting on Inventory Results

This topic applies to all forms of gathered inventory:

- All forms of FlexNet inventory gathering, including all the cases covered in this document

- Inventory imported from third-party tools, such as Microsoft Endpoint Configuration Manager (previously Microsoft SCCM), IBM ILMT, and others
- Inventory imported in spreadsheets or CSV files.

Computer details imported from software and hardware inventory are displayed in the web interface of FlexNet Manager Suite under **Discovery & Inventory > All Inventory**. Here you may choose whether to:

- Manage each device as an asset.
- Mark the device as `Ignored` (unmanaged). At a later time, you may choose to begin managing any ignored computers.

These cases and others are explained below.

Managed Assets

You may manage the following device types as assets:

- Computers
- Mobile devices
- VM Hosts.

Virtual machines, VDI templates, and remote devices are not tangible assets (remote devices are created when the only inventory available is usage or access control list data from a Citrix environment, when it assumed that, for example, a user's home computer that cannot be inventoried has used a virtualized application).

When your goal is to link devices to asset records, it may be easier to start from **Discovery & Inventory > Inventory without Assets**. In any inventory list, simply select one or more devices, and click **Create an asset** (see the online help for more information).

Once you choose to manage a device as an asset, you are then able to perform all of the normal hardware asset operations:

- Assign ownership of the asset in your enterprise (as you can also assign ownership of computers that are not assets)
- Associate the asset with purchase orders and contracts (including lease contracts) and monitor expiry of lease contracts
- Monitor warranty details
- Allocate licenses to the computer (licenses are also managed on devices that are not assets)
- Have FlexNet Manager Suite automatically flag changes made to the device’s configuration
- Receive alerts for any assets that stop reporting their configuration details during compliance imports.

Ignored Computers

Inventory devices that you choose to ignore are those that you do not wish to manage, at least for the present time. The records are not deleted, but these devices become 'inactive' in the following ways:

- Ignored devices are not visible in most inventory listings, including **Inventory Issues**, **Out-Of-Date Inventory**, **Inventory without Assets**, and **Active Inventory** (but of course, they are available in **Ignored Inventory**)

- Changes to the configuration of ignored devices are not flagged
- If an ignored device stops reporting during inventory imports, the missing device does not appear in the **Out-Of-Date Inventory** list
- Applications installed on ignored computers are not counted in license compliance calculations.

Tip: *You cannot ignore an inventory device that is linked to an asset record. If you need to do this, first select the device in a listing, and choose* ***Remove link****.*

At any time, you can choose to begin managing an ignored device as a hardware asset. Simply select the device in **Discovery & Inventory > Ignored Inventory**, and click **Activate**.

Duplicate serial numbers

FlexNet Manager Suite does not rely on serial numbers alone to differentiate between inventory device records. For example, other attributes used may include:

- The external ID, the identifier for the inventory item in the source data (for example, the numeric `MachineID` returned from Microsoft Endpoint Configuration Manager (previously Microsoft SCCM), or the numeric `ComputerID` assigned in the internal inventory database of FlexNet inventory data)
- The computer type, manufacturer, and model
- For virtualization, the type of virtual machine, and (when applicable) a partition ID

and so on.

However, it happens that inventory records may be different in all other values, but have matching (duplicate) serial numbers. There are a few common causes for inventory correctly containing multiple devices with a common serial number:

- Computer manufacturers do not always assign unique serial numbers, so that it is quite possible to get duplicate serial numbers from quite separate machines (particularly when these were bought in a batch).
- A computer may have been partitioned, and the inventory tool may return the common serial number of the underlying host for separate partitions, producing multiple records with the same serial number.
- Computers that have been rebuilt may, for a time, produce two different inventory records with the same serial number — one from before the rebuild that has not yet gone stale and been removed by the inventory tool (the last inventory date on this record is typically old), and a second from after the rebuild. Typically these will have the same serial number; but the rebuilt computer may have a different name and be deployed to a new domain. If, as commonly happens, the inventory tool does not recognize the new computer as a rebuild, the two records will also have different external IDs from the inventory tool. This means that nothing matches except the serial numbers.

You can identify inventory devices with duplicate serial numbers as follows:

1. Navigate to your preferred listing of inventory devices (for example, **Discovery & Inventory > All Inventory**).
2. Clear any existing filter, and in the simple filter control (top left, near the page heading):
   a. Click **Add filter**.
   b. In the left-hand option list, choose `Alert`.

c. After a moment, in the right-hand option list, choose `Serial number is not unique`.

d. Click the blue check (tick) icon to commit this filter. After a moment, only devices with duplicate serial numbers are shown (hover over any of the red alert bubbles to confirm this by reviewing the tooltip text).

3. Click the **Serial number** column header to sort the listing by serial number. This groups the matched sets together.

You can now investigate the duplicates. It may be possible, for example, to ignore old device records of machines that have been rebuilt (although best practice is to have them cleaned out of the inventory source data).

Common: Resolving Inventory Records

This topic applies to all forms of gathered inventory:

- All forms of FlexNet inventory gathering, including all the cases covered in this document
- Inventory imported from third-party tools, such as Microsoft Endpoint Configuration Manager (previously Microsoft SCCM), IBM ILMT, and others
- Inventory imported in spreadsheets or CSV files.

One of the primary tasks performed by FlexNet Manager Suite when it imports inventory is to resolve multiple inventory records that all represent the same device in your environment. (It's easy to imagine this for physical devices like a workstation; but it also applies to virtual machines as well.)

The simplest example of multiple records is when inventory was collected from my workstation last week, and new inventory was collected again this week. Clearly the new inventory record has to be matched to the previous inventory record, so that values can be updated. Or if there is no match with a previous record, a new inventory device record must be created.

However, there may be other reasons why multiple records need resolution, even within a single inventory import. For example:

- You have multiple tools collecting inventory (say, ADDM and ILMT), and they overlap so that some devices appear in both sources. How does the inventory import process decide when two or more records actually describe the same device?
- In a variation on that theme, you may use one tool (say, Microsoft Endpoint Configuration Manager (previously Microsoft SCCM)) as the primary source of inventory data, but deploy FlexNet inventory agent to specific devices to augment details about Microsoft SQL Server. How do these two sources get combined?
- You may have deployed the FlexNet inventory agent to multiple devices which have the same host name and domain name. How do these inventories get handled? These duplicates may arise:
  - On disaster recovery or failover environments, where the same names were used as for the production systems
  - On cloned virtual machines, where the cloning process did not include updating the machine's name
  - With public cloud instances created from a single image with a fixed host name.
- You deployed FlexNet inventory agent to some devices (in the Agent third-party deployment case), but because these devices are not identified in any target for adoption, the relevant inventory beacon is also collecting Zero-footprint inventory. Will these records clash somehow, or be resolved?

You may have additional reasons to seek a deep understanding of the process, such as:

- You want to understand why some devices visible in your Microsoft Endpoint Configuration Manager (previously Microsoft SCCM) listing are not visible in FlexNet Manager Suite, even though Microsoft Endpoint Configuration Manager is your main inventory tool.
- You are creating a custom inventory adapter, and want to know which fields must be imported for successful socialization with the inventory records already existing in FlexNet Manager Suite.

There are three parts to the question of data resolution:

- Ensuring that the inventory returns from distinct installations of FlexNet inventory agent are not merged. For details, see Common: Ensuring Distinct Inventory.
- Determining when multiple records apply to the same inventory device. For full details, see Common: Identifying Related Inventory.
- Once a set of imported records is known to apply to the same device, how do we select which data values to use from which record? See Common: Choosing Values from Multiple Inventory Records.

Common: Ensuring Distinct Inventory

While the issue identified here applies to all inventory gathered by FlexNet inventory agent, the solution applies only to:

- Adopted case, where the FlexNet inventory agent is locally installed and managed by policy
- Agent third-party deployment, which results in the same policy-based local agent
- Zero-footprint for UNIX-like platforms, where the `ndtrack.ini` configuration file can be co-located on the inventory beacon with the `ndtrack.sh` script
- FlexNet Inventory Scanner case for UNIX-like platforms, where (once again) the `ndtrack.ini` configuration file can be co-located with the `ndtrack.sh` script.

On a Windows device, the last two cases are supportable if your infrastructure allows you to run a custom command line for the inventory component (`ndtrack`).

All device inventory gathered through the FlexNet inventory agent is identified in either of the following ways depending on the version of the FlexNet inventory agent being used:

- If the version is 2019 R2 or later, the device inventory is identified by a unique Agent ID that is created on the device by the FlexNet inventory agent.
- For versions prior to 2019 R2, the device inventory is identified by the computer name and domain name of the device.

In the scenario where the computer name and domain name of the device are used to identify the device inventory, this is normally enough to uniquely identify machines, but there are scenarios where these keys can become non-unique. For example:

- You have a disaster recovery or failover environment within your enterprise which hosts machines with the same computer and domain names as are used in your production environment.
- You have virtual machines or public cloud computer instances (for example, AWS EC2 instances) which are spun up

on demand from a primary image (Amazon Machine Image, or AMI). These machines only exist for a limited time to perform some compute-intensive task, and so do not normally require distinct computer names. Therefore each instance automatically adopts its computer name from the default base image: that is, each instance adopts the same computer name. Since they commonly also report the same domain name, the records appear as duplicates.

To distinguish these machines as unique computers, FlexNet inventory agent allows you to customize the computer name and domain name reported in inventory. This is controlled by the following preferences for FlexNet inventory agent:

- `ComputerDomain` for the domain name of the device (see ComputerDomain)
- `MachineID` for the name of the device (see MachineID).

Overriding these settings is only possible either through

- Configuration for FlexNet inventory agent:
  - In the registry for Windows platforms
  - In the `config.ini` pseudo-registry for UNIX-like platforms with the locally-installed FlexNet inventory agent (Adopted or Agent third-party deployment cases)
  - In the equivalent `ndtrack.ini` configuration file for UNIX-like platforms (Zero-footprint or FlexNet Inventory Scanner cases)
- The command line, where the situation allows for a custom command line that can include, for example:

```
-o MachineID=UniqueDeviceName
```

Note: *Downgrading the version of the FlexNet inventory agent to a version prior to 2019 R2 may result in additional inventory device records being created as inventory from older FlexNet inventory agent versions will not be matched with inventory devices that have an Agent ID.*

Tip: *When the FlexNet inventory agent or FlexNet Inventory Scanner are deployed as part of a base virtual machine, or within a public cloud compute image from which instances are dynamically instantiated, ensure that the VM or image start-up scripts appropriately configure these preferences to give a unique device identification.*

See the individual preference topics for further details.

Common: Identifying Related Inventory

This topic applies to all forms of gathered inventory:

- All forms of FlexNet inventory gathering, including all the cases covered in this reference
- Inventory imported from third-party tools, such as Microsoft Endpoint Configuration Manager (previously Microsoft SCCM), IBM ILMT, and others.

Before gathered data sets can be merged into distinct inventory records, the group (or set) of imported records that relate to a single device must be identified. This topic gives considerable detail about the process of matching records into sets that relate to a single device.

The process of grouping incoming inventory records into matched sets, such that each set relates to a single unique

device, is controlled by an XML file (from which the examples below are taken). The process is as follows:

1. All incoming inventory records held in the staging tables are copied into memory for faster processing.
2. An ordered list of assessments (called "Matchers") is applied to the incoming inventory records. Each Matcher is applied in turn:
   a. A Matcher preselects only those records that fit a list of conditions. Whenever a list of multiple conditions is present, every condition in the list must be matched.

      (Strictly speaking, conditions are an optional part of a Matcher, but they are ubiquitous. If ever there were no conditions, the subsequent tests in the Matcher would be applied to every inventory record currently left in the memory array.)

      Each Condition specifies an inventory Property, a testing Method, and one or more values used as the test cases. Those records that meet all the conditions in the list are used for further processing in this Matcher, while other records not meeting the current list of conditions are left aside for later reconsideration in other Matchers. In short, the list of conditions filters down the set of incoming inventory records to be assessed in the current Matcher.

      For example:

      ```
      <Condition Property="ComputerType" Method="InRange" Value="Computer,
      VMHost"/>
      ```

      This condition admits only computers and VM hosts for possible matching, and excludes VMs, remote devices, mobile devices, and VDI templates.

   b. The Matcher then tests the preselected processing candidates using a list of one or more rules. Every rule in the Matcher's list must be satisfied.

      Each rule specifies that a named inventory Property must match across inventory records, using a defined matching Method (the methods include `equal`, `not equal`, `like`, `set`, and `not set`, with the default being `equal`). Pairs (or sets) of devices that pass all the rules are considered matched, and those that fail any rule in the list are set aside for later reconsideration in other Matchers.
      For example:

      ```
      <Rule Property="Manufacturer"/>
      <Rule Property="ModelNo"/>
      <Rule Property="FirmwareSerialNumber"/>
      ```

      This list of rules means that when each of these properties in turn has a common value across candidate records (because "equal" is the default when no Method is specified), inventory records from the preselected processing candidates are considered "matched", so that they refer to a single device.

   c. Any group of records that have been matched by this Matcher are now known to refer to a single device. They are removed from the testing pool, and are not subjected to any further assessment by other Matchers. Each matched group (or set) is queued up for import into the compliance database.
3. After all Matchers have been processed in turn, the unmatched records left in the testing pool are known to be individual records each applying to a unique device, and these are now similarly queued for import into the compliance database.
4. The queue of matched sets (including the sets with just a single member) are now tested for matching against the existing records in the compliance database. The exact same tests used to bunch up the incoming inventory

records are now applied to align the matched sets with existing records.

- If any Matcher in the XML file finds that an incoming set is matched to an existing record, that existing record is updated with values from the incoming set. The constraints about Primary source and latest inventory date are used to select which of the incoming set of records is used to update each property (see Common: Acting on Inventory Results).
- If all Matchers fail to connect an incoming set with an existing record, a new inventory device record is created. Once again, properties are taken first from the Primary inventory source with gaps filled from secondary inventory sources; and competition is resolved by using the most recent inventory if there are multiples from the same priority source(s).

The tests used to match sets of inventory records to individual devices are listed in the following table. The Matchers run in order from top to bottom in this table. In each case, every Condition must be satisfied before an inventory record is included in an individual assessment; and then every Rule must be satisfied before two (or more) records are taken as matched, applying to a single device. Both Conditions and Rules assess Properties from (firstly) the incoming inventory records, which in the database have been staged in the appropriate staging tables:

- **`ImportedComputer`** table
- **`ImportedVirtualMachine`** table.

Thereafter, the incoming matched sets are compared against existing compliance records using the same set of Matchers. The database schema is described in *FlexNet Manager Suite Schema Reference*.

Table 1: Matchers

Candidates (Conditions)	Assessment (Rules)
1. Matcher for VMware ESX Servers via ComputerName and DomainID	
• `InstanceCloudID` is null in either or both of the records being compared • `OperatingSystem` contains vmware (see note 1) • `ComputerName` is set • `ComplianceDomainID` is set	Records have identical values for each of: • `ComputerName` • `ComplianceDomainID`
2. Matcher for physical computer with manufacturer, model, firmware serial number and host ID	

Candidates (Conditions)	Assessment (Rules)
• InstanceCloudID is null in either or both of the records being compared • VMType is not set • Manufacturer is set • ModelNo is set • HostID is set • FirmwareSerialNumber is set	Records have identical values for each of: • HostID • FirmwareSerialNumber • ModelNo • Manufacturer
3. Matcher for physical computer with manufacturer, model and firmware serial number when no host ID is available	
• InstanceCloudID is null in either or both of the records being compared • VMType is not set • Manufacturer is set • ModelNo is set • HostID is not set • FirmwareSerialNumber is set	Records have identical values for each of: • FirmwareSerialNumber • ModelNo • Manufacturer
4. Matcher for physical computer with manufacturer, host type, firmware serial number and host ID	
• InstanceCloudID is null in either or both of the records being compared • VMType is not set • Manufacturer is set • HostID is set • HostType is set • FirmwareSerialNumber is set	Records have identical values for each of: • HostID • FirmwareSerialNumber • HostType • Manufacturer
5. Matcher for physical computer with manufacturer, host type and firmware serial number when no host ID is available	

Candidates (Conditions)	Assessment (Rules)
• `InstanceCloudID` is null in either or both of the records being compared • `VMType` is not set • `Manufacturer` is set • `HostID` is not set • `HostType` is set • `FirmwareSerialNumber` is set	Records have identical values for each of: • `FirmwareSerialNumber` • `HostType` • `Manufacturer`
6. Matcher for physical computer with machine ID	
• `InstanceCloudID` is null in either or both of the records being compared • `VMType` is not set • `MachineID` is set	Records have identical: • `MachineID`
7. Matcher for physical computer with manufacturer, host ID and computer name	
• `InstanceCloudID` is null in either or both of the records being compared • `VMType` is not set • `Manufacturer` is set • `HostID` is set • `FirmwareSerialNumber` is set	Records have identical values for each of: • `ComputerName` • `HostID` • `Manufacturer`
8. Matcher for vPar, LPAR and Zone with Partition ID	
• `InstanceCloudID` is null in either or both of the records being compared • `VMType` is one of vPar, LPAR, or Zone • `PartitionID` is set	Records have identical values for each of: • `PartitionID` • `VMType`
9. Matcher for LPAR with firmware serial number and partition number	

Candidates (Conditions)	Assessment (Rules)
• `InstanceCloudID` is null in either or both of the records being compared • `VMType` is LPAR • `FirmwareSerialNumber` is set • `PartitionNumber` is set	Records have identical values for each of: • `PartitionNumber` • `FirmwareSerialNumber`
10. Matcher for LPAR with Machine ID and partition number	
• `InstanceCloudID` is null in either or both of the records being compared • `VMType` is LPAR • `MachineID` is set • `PartitionNumber` is set	Records have identical values for each of: • `PartitionNumber` • `MachineID`
11. Matcher for vPar, nPar and LPAR with machine ID and partition name	
• `InstanceCloudID` is null in either or both of the records being compared • `VMType` is one of vPar, nPar, or LPAR • `VMName` is set • `MachineID` is set	Records have identical values for each of: • `VMName` • `VMType` • `MachineID`
12. Matcher for vPar, nPar and LPAR with partition name and firmware serial number	
• `InstanceCloudID` is null in either or both of the records being compared • `VMType` is one of vPar, nPar, or LPAR • `FirmwareSerialNumber` is set • `VMName` is set	Records have identical values for each of: • `VMName` • `VMType` • `FirmwareSerialNumber`
13. Matcher for Zones with host ID and partition name	

Candidates (Conditions)	Assessment (Rules)
• InstanceCloudID is null in either or both of the records being compared • VMType is Zone • HostID is set • VMName is set	Records have identical values for each of: • VMName • HostID
14. Matcher for computers using HostID, HostIdentifyingNumber, HostType and Manufacturer	
• InstanceCloudID is null in either or both of the records being compared • HostID is set • HostIdentifyingNumber is set • HostType is set • Manufacturer is set	Records have identical values for each of: • HostID • HostIdentifyingNumber • HostType • Manufacturer
15. Matcher for computers using HostIdentifyingNumber, HostType and Manufacturer when HostID not provided	
• InstanceCloudID is null in either or both of the records being compared • HostID is not set • HostIdentifyingNumber is set • HostType is set • Manufacturer is set	Records have identical values for each of: • HostIdentifyingNumber • HostType • Manufacturer
16. Matcher for computers using HostID, HostIdentifyingNumber and HostType when Manufacturer not provided	
• InstanceCloudID is null in either or both of the records being compared • HostID is set • HostIdentifyingNumber is set • HostType is set • Manufacturer is null in either or both of the records being compared	Records have identical values for each of: • HostID • HostIdentifyingNumber • HostType

Candidates (Conditions)	Assessment (Rules)
17. Matcher for computers using HostIdentifyingNumber and HostType when HostID and Manufacturer not provided	
• `InstanceCloudID` is null in either or both of the records being compared • `HostID` is not set • `HostIdentifyingNumber` is set • `HostType` is set • `Manufacturer` is null in either or both of the records being compared	Records have identical values for each of: • `HostIdentifyingNumber` • `HostType`
18. Matcher for computers using HostID, HostIdentifyingNumber and Manufacturer when HostType not provided	
• `InstanceCloudID` is null in either or both of the records being compared • `HostID` is set • `HostIdentifyingNumber` is set • `HostType` is null in either or both of the records being compared • `Manufacturer` is set	Records have identical values for each of: • `HostID` • `HostIdentifyingNumber` • `Manufacturer`
19. Matcher for computers using HostIdentifyingNumber and Manufacturer when HostID and HostType not provided	
• `InstanceCloudID` is null in either or both of the records being compared • `HostID` is not set • `HostIdentifyingNumber` is set • `HostType` is null in either or both of the records being compared • `Manufacturer` is set	Records have identical values for each of: • `HostIdentifyingNumber` • `Manufacturer`
20. Matcher for computers using HostID and HostIdentifyingNumber when HostType and Manufacturer are not provided	

Candidates (Conditions)	Assessment (Rules)
• InstanceCloudID is null in either or both of the records being compared • HostID is set • HostIdentifyingNumber is set • HostType is null in either or both of the records being compared • Manufacturer is null in either or both of the records being compared	Records have identical values for each of: • HostID • HostIdentifyingNumber
21. Matcher for computers using HostIdentifyingNumber when HostID, HostType and Manufacturer are not provided	
• InstanceCloudID is null in either or both of the records being compared • HostID is not set • HostIdentifyingNumber is set • HostType is null in either or both of the records being compared • Manufacturer is null in either or both of the records being compared	Records have identical values for each of: • HostIdentifyingNumber
22. Matcher for computers using the serial number and computer name	
• InstanceCloudID is null in either or both of the records being compared • UntrustedSerialNo is false • SerialNo is set • ComputerName is set	Records have identical values for each of: • SerialNo • ComputerName
23. Matcher for computers using the agent ID which IBM ILMT uses for tracking operating environments	
• InstanceCloudID is null in either or both of the records being compared • ILMTAgentID is set	Records have identical values for each of: • ILMTAgentID
24. Matcher for computers using InstanceCloudID from the ImportedCloudServiceInstance table	

Candidates (Conditions)	Assessment (Rules)
• `InstanceCloudID` is set	Records have identical values for each of: • `CloudServiceProviderID` • `InstanceCloudID`
25. Matcher for incomplete computers using computer name and domain details	
• The `ObjectType` is `Incomplete` (see note 2) • `ComputerName` is set • `ComplianceDomainID` is set	Records have identical values for each of: • `ComputerName` • `ComplianceDomainID`
26. Matcher for incomplete computers using computer name where no domain is available	
• The `ObjectType` is `Incomplete` (see note 2) • `ComputerName` is set • `ComplianceDomainID` is null in either or both of the records being compared	Records have identical values for: • `ComputerName`
27. Matcher for unmatched computers without hardware details using computer name and domain details	
• The `ObjectType` is `Unmatched` (see note 2) • `InstanceCloudID` is null in either or both of the records being compared • `ComputerName` is set • `ComplianceDomainID` is set • `SerialNo` is not set	Records have identical values for each of: • `ComputerName` • `ComplianceDomainID`
28. Matcher for unmatched computers without hardware details using computer name and domain details along with UntrustedSerialNo	
• The `ObjectType` is `Unmatched` (see note 2) • `InstanceCloudID` is null in either or both of the records being compared • `UntrustedSerialNo` is true • `ComputerName` is set • `ComplianceDomainID` is set	Records have identical values for each of: • `ComputerName` • `ComplianceDomainID`

Candidates (Conditions)	Assessment (Rules)
29. Matcher for unmatched computers using computer name where no domain is available	
• The `ObjectType` is `Unmatched` (see note 2) • `InstanceCloudID` is null in either or both of the records being compared • `ComputerName` is set • `ComplianceDomainID` is null in either or both of the records being compared	Records have identical values for: • `ComputerName`

Note:

1. *For Condition checks, the method "contains" tests whether the value of the property under test includes the test string in any position.*
2. *Here, `ObjectType` is an intermediate value calculated during import and not saved to the compliance database. It may be `Complete` (default), `Incomplete`, or `Unmatched`.*

Tip: *The XML file of Matchers is located on your batch server (or, in smaller implementations, on whichever server hosts this functionality, such as your processing server or your single application server), in `%ProgramData%\Flexera Software\Compliance\ImportProcedures\Inventory\Matcher\Computer.xml`. Do not edit this file, as any changes may be over-written in the next product upgrade.*

While you should not edit the factory-supplied `Computer.xml` file, there is a supported method to integrate a custom Matcher into the process described above. To customize a Matcher, you will need to create your own `Computer.xml` with the root Matchers element, and a corresponding `matcher.config` file.

Creating the Computer.xml file

1. Create your own `Computer.xml` file with the root `Matchers` element:

```
<?xml version="1.0" encoding="utf-8"?>
<Matchers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

</Matchers>
```

2. In the gap, insert your custom `Matcher` element, modeled on the factory-supplied ones in the standard (unchanged) file.
3. Set the `Order` attribute of your Matcher so that it will fall in the correct place between the standard Matchers. For example, if you wish to override the current Matcher with `Order="30"`, set your `Order="25"` so that your custom Matcher runs first, "stealing" the candidates it needs and removing matched sets from the pool before the standard `Order="30"` Matcher runs. (You do not need to remove the standard Matcher, since if it has no suitable candidates to match, it runs with no net effect, and little impact on performance.) The standard Matchers are ordered with numbering gaps that accommodate 'insertions' like this.

4. Save your `Computer.xml` file separately in `%ProgramData%\Flexera Software\Compliance\ImportProcedures\CustomInventory\Matcher\Computer.xml`.

Both the standard `Computer.xml` file and your customized one are read from their separate locations, their Matchers are merged into a single ordered list, and executed in the process as described above.

Creating the matcher.config file

1. Create the `matcher.config` file using the following code. **Note:** The code can be used as below without modification.

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration>
    <Matchers>
        <Matcher FileName="Computer.xml" ProcedureName="Computer" />
        <Matcher FileName="Cluster.xml" ProcedureName="Cluster" />
    </Matchers>
</Configuration>
```

2. Save the config file in `%ProgramData%\Flexera Software\Compliance\ImportProcedures\CustomInventory\Matcher\matcher.config`.

Common: Choosing Values from Multiple Inventory Records

This topic applies to all forms of gathered inventory:

- All forms of FlexNet inventory gathering, including all the cases covered in this document
- Inventory imported from third-party tools, such as Microsoft Endpoint Configuration Manager (previously Microsoft SCCM), IBM ILMT, and others
- Inventory imported in spreadsheets or CSV files.

Once sets of imported inventory records that relate to a single device have been identified (see Common: Identifying Related Inventory), it is still necessary to decide, if the records have different values for any properties, which value should be used to update the "golden record" in the compliance database. Since it is possible that one property will be selected from imported record A and a different property from imported record B, this process is called data *merging*.

The process of data merging is relatively straight-forward:

- All data from the primary inventory source is used. If there are two data sets from the primary source (for example, last week's inventory and this week's inventory), values from the record with the most recent date/time stamp are used. (The primary data source is selected through the system menu (▼ in the top right corner) select **Data Inputs** and choose the **Inventory Data** tab.)
- For any individual property values missing from the primary source record:
  - If a missing property appears in any single secondary source, it is inserted to augment the data from the primary source (however, existing values taken from the primary inventory source are never modified)

- If missing properties appear in multiple secondary sources, values from the record with the most recent date/time stamp are used (secondary sources cannot be prioritized relative to one another by any means other than the inventory collection date).

In these ways, information gathered from primary and secondary inventory sources is merged to produce a single record in which data from the primary source is always dominant.

The fact that values are merged individually means that data from a previous inventory collection may still "show through" in an updated record. For example, imagine this scenario:

- ADDM is used as your primary inventory source, and its inventory is imported daily.
- FlexNet inventory agent is installed on selected long-running servers, and inventory is taken weekly. This inventory contains values not available in ADDM.

On Monday, inventory of an AIX system is imported from both sources. The merging process (described above) inserts into the primary record (from ADDM) the missing value of the host type (`8202`, collected by FlexNet inventory agent).
On Tuesday, new ADDM inventory is imported, and the fields collected by ADDM are updated. Because there is no new inventory from the secondary sources on this day, the value of the host type previously collected is left unchanged. The inventory device record displayed in FlexNet Manager Suite remains complete.

8

Selecting Inventory Beacons

This chapter describes how the FlexNet inventory agent (or its components) locally installed on a target inventory device determines which inventory beacon to use for a download or upload operation, and how you can choose between a combination of supplied options that the components may use to make this choice. It includes:

- Background information on how the process works (see Overview)
- Where settings are saved, and how you may update them (see Saving the Configuration)
- The range of preference settings that affect the process of prioritizing inventory beacons (see Prioritizing Inventory Beacons)
- Full details of each of the supported algorithms that an inventory beacon may use in the prioritizing process (see Supplied Algorithms and following topics).

Overview

In FlexNet Manager Suite, inventory beacons are used as staging posts between the application server and target inventory devices. Inventory devices retrieve data from the *distribution locations* on inventory beacons, and upload data to *reporting locations* on inventory beacons. Notice that all communications (regardless of the perceived 'direction') are initiated by the installed FlexNet inventory agent. Each of these processes is atomic, meaning that the agent components may reassess which inventory beacon to use for each individual operation.

Details of *accessible* distribution locations and reporting locations in your inventory beacon hierarchy are stored on each target inventory device. These locations are "accessible" when the target inventory device can successfully communicate with the inventory beacon. In the main, this means that the inventory beacon is configured for anonymous authentication; although an inventory beacon using Windows Basic Authentication may also be included, if it is known that the target inventory device already possesses the credentials to use for that inventory beacon. The mechanism works like this:

- A source list of inventory beacons configured for anonymous authentication is automatically maintained on the central application server.
- The source list is distributed to all inventory beacons as a resource, and updated (when needed) as part of the inventory beacon policy update.
- An installed FlexNet inventory agent chooses an inventory beacon and requests an update to its device policy. It may

happen that the FlexNet inventory agent makes this request to its bootstrap inventory beacon (the one it contacted first after installation), for which it may have been provided credentials as part of its installation.

- In response, the inventory beacon prepares a list of available inventory beacons, making it ready to download to the requesting target inventory device. If this inventory beacon itself uses Windows Basic Authentication, it knows that the requesting inventory device possesses the appropriate credentials (just used to make the policy update request), and therefore inserts itself into the list of available inventory beacons. This is the only standard way that an inventory beacon that does *not* accept anonymous authentication is included in this "failover list" of available inventory beacons.

Tip: *To maintain backward compatibility with legacy versions of the FlexNet inventory agent, the failover list references server names for the inventory beacons, rather than IP addresses which might be in the IPv6 address family (not supported by legacy versions of FlexNet inventory agent).*

- When the policy update for the target inventory device includes a changed list of available inventory beacons, the inventory device saves all the details in its registry (on Microsoft Windows) or its registry-equivalent `config.ini` file (on UNIX-like platforms).

As some inventory beacons are better suited to a given inventory device than others, the inventory device can prioritize the list of inventory beacons. (You may also override this capability by setting a fixed priority.) To determine which inventory beacon to use for upload and download activities, inventory devices use a set of rules, called algorithms, to assign priorities to each inventory beacon. Thereafter the inventory beacon with the highest priority (lowest numeric value) is used. This is commonly referred to as determining the *closest* inventory beacon, although the inventory beacon may not be the one physically closest to the inventory device.

In the event that the highest priority inventory beacon cannot be used (for example, because of network problems), the inventory beacon with the next highest priority is used, and so on until the download or upload activity can be performed. It is also possible to configure the number of, and period between, attempts to connect to each inventory beacon before switching to the next one in its prioritized failover list (see NetworkRetries and NetworkRetryPeriodIncrement).

Saving the Configuration

Details of all download locations and reporting locations in your hierarchy of inventory beacons are stored on each target inventory device that has a locally-installed FlexNet inventory agent (or core components of it).

- On inventory devices running Microsoft Windows, details are saved in the Windows registry
- On UNIX-like devices, they are stored in the `/var/opt/managesoft/etc/config.ini` file.

For more information about the root registry keys, see [Registry] Explained.

The details stored in the `[Registry]` about download and upload locations include folder and host names, and port numbers. This information is stored in the `DownloadSettings` and `UploadSettings` preferences (for details, see DownloadSettings and UploadSettings).

Tip: *By default, the "bootstrap" inventory beacon used in adoption of the inventory device is assigned a low priority (100), so that it is naturally overridden by other inventory beacons identified in downloaded device policy.*

You can configure other registry settings to adjust the way that target inventory devices assign priorities to inventory

beacons (summarized in Prioritizing Inventory Beacons).

The method for prioritizing inventory beacons used by each inventory device is declared in the `SelectorAlgorithm` preference (see SelectorAlgorithm). For software and hardware inventory management, the default value (`MgsRandom;MgsPing;MgsADSiteMatch` — the last algorithm does not apply to UNIX-like devices) is normally adequate. Because of the relatively simple download and upload requirements for inventory management, there is no way in the presentation of the system to modify the selected algorithms. If you wish to modify `SelectorAlgorithm` or other preferences impacting the selection of inventory beacons by inventory devices, choose one of the following:

- Manually edit the preferences as documented, in the case where you only wish to modify a very few target inventory devices
- Use your preferred administrative tool to deploy registry settings to target Windows devices, or a customized `config.ini` to UNIX-like devices.
- On UNIX-like platforms only, you can use (or script) the import feature of the `mgsconfig` tool:

  1. Create a temporary file containing only the desired change, such as:

     ```
     [ManageSoft\NetSelector\CurrentVersion]
      SelectorAlgorithm=MgsSubnetMatch(,false)
     ```

  2. Deploy this temporary file to UNIX-like target inventory devices (for example, deploy to `/var/tmp/tempconfig.ini`).
  3. Run the following command on the target device to import the configuration change (substituting your actual path and file name for the temporary file):

     ```
     /opt/managesoft/bin/mgsconfig -i /var/tmp/tempconfig.ini
     ```

  4. Remove the temporary file.

  This method merges the change into the `config.ini` file without disturbing other custom settings that may have been configured for various target inventory devices.

Prioritizing Inventory Beacons

A number of registry settings affect the way in which a target inventory device manages its failover list of inventory beacons. The first two are attached to each record of an upload or download location, and the others are generalized settings controlling the overall process:

- `AutoPriority` determines whether this location can take part in the process of determining priorities (see AutoPriority). When `AutoPriority` is `true`, the target inventory device uses the approved algorithm(s) to determine the priority of this location at the time of upload or download. This is the recommended behavior, as it optimizes system performance over time, despite variations in network configuration. When `AutoPriority` is `false`, the value of the `Priority` registry key declares the inventory beacon's fixed priority.
- `Priority` may be manually set to a fixed value for use when `AutoPriority` is `false` (see Priority). (In the normal case, when `AutoPriority` is `true`, the value of `Priority` is calculated dynamically.)
- `SelectorAlgorithm` specified which one (or more) of the available algorithm(s) the inventory device should use

(and in what order the algorithms should be applied) when determining the prioritized failover list of inventory beacons (see SelectorAlgorithm).

- `HighestPriority` and `LowestPriority` set the upper and lower bounds for the normalized range of priorities on which the inventory device settles. For example, the inventory device may initially calculate a set of values as 34, 96, 12, and 242. It then normalizes these figures to order them with the range set in `HighestPriority` and `LowestPriority`. For example, if these had the respective values of `1` and `5`, the inventory beacons now receive the values 2, 3, 1, and 4 (in matching order). It is only the ordering, and not any measure of deviation, that matters in the selection process. (Normally, ignore these settings, but if you really need them, details are in HighestPriority and LowestPriority.)

Using a Single Inventory Beacon

There are some scenarios where you may want to force a set of target inventory devices to get policy updates from, and upload discovery and inventory files to, a specific inventory beacon. This is contrary to the design of FlexNet Manager Suite, which provides for each policy-driven, self-managing device to choose the optimum path in current conditions for its downloads and uploads. If a preferred inventory beacon is overloaded, or even completely unavailable, the FlexNet inventory agent is able to find the current best fit to ensure uninterrupted operations.

Nevertheless, if your situation genuinely demands a different set of guiding principles, you can use the available preferences to bend the system to suit. Locking a target inventory device into a specific inventory beacon requires three main changes:

- Configuring an upload location and download location for the device to use, in such a way that the settings are no longer updated by downloaded device policy
- Setting a fixed, top priority for the inventory beacon the device is to use
- Changing the algorithm used for selecting inventory beacons, so as to prevent failover to any server other than the specified inventory beacon.

Keep in mind that, in deciding to defeat automated failover to the best available inventory beacon, you are taking responsibility for manually managing connectivity for your inventory devices. If, for example, you move an inventory beacon or take it out of service for a period of maintenance, you risk loss of inventory from all devices reporting to this inventory beacon, with potential negative impacts on your reported license position as your data decays over time. For these reasons, the following approach should be used only when truly necessary.

You will also need your own method of deploying settings for the registry on Windows target devices, or deploying a customized `config.ini` file to devices with UNIX-like operating systems, as discussed in Saving the Configuration. The values in this package are unique for each inventory beacon, so you may need a range of packages to suit the number of inventory beacons that are to be individually targeted. For example, if you are specifying an inventory beacon per region to manage inventory devices within that region, then each region needs is own custom package of registry settings.

To prepare registry settings for locked-down inventory devices:

1. Configure `SelectorAlgorithm` to prevent failover to any inventory beacons downloaded in policy.

   Policy downloaded to the inventory device lists a pool of inventory beacons that it may prioritize for failover. The following setting makes every one of these downloaded inventory beacons invalid for prioritization:

```
[Registry]\ManageSoft\NetSelector\CurrentVersion:
    MgsNameMatch(20,,true)
```

This algorithm matches the first 20 characters of the host name of the inventory beacon with the name of the local inventory device. There is no limit set on the number of inventory beacons that will be tested; and every inventory beacon which fails the name comparison has its priority set to `invalid`, so that it is excluded from the failover list. Effectively, this disables the entire downloaded failover list.

Tip: *Normally, the default value for `SelectorAlgorithm` is set at installation of the FlexNet inventory agent, and is not subject to updates by device policy. This means that your new custom setting will not be updated by future policy updates to each target device.*

2. Define custom settings for the upload and download locations used by the inventory device(s).

   Do not modify the upload/download settings already registered for the inventory beacons in the failover list, as these are updated as required with each device policy update. Instead, copy a pair of settings (or create new ones) with a custom name for the inventory beacon instead of the GUIDs used within the failover lists. Since each inventory device tests *all* the inventory beacons listed in these upload/download registry keys, your new instance is evaluated (and, given the previous step, is the only one that can be used). For more information, see DownloadSettings and UploadSettings. Notice that:

   - The `Host` value may be the host name, the fully-qualified domain name, or the IPv4 or IPv6 address
   - If you wish to use Windows Authentication (user name and password values), the `Password` value must be encrypted: copy the existing encrypted password already saved for the intended inventory beacon, and paste into the settings for distribution. (The example shown below is for anonymous authentication.)

   Define keys in the following manner, using your custom values rather than the placeholders shown here:

```
[Registry]\ManageSoft\Common\
DownloadSettings\SpecialInventoryBeaconIDWithoutWhiteSpace
      Protocol=http
      Name=BeaconFriendlyName Download Location
      Directory=ManageSoftDL
      Host=BeaconHost
      Port=80
      User=
      Password=
      Priority=1
      AutoPriority=false
[Registry]\ManageSoft\Common\
UploadSettings\SpecialInventoryBeaconIDWithoutWhiteSpace
      Protocol=http
      Name=BeaconFriendlyName Reporting Location
      Directory=ManageSoftRL
      Host=BeaconHost
      Port=80
      User=
      Password=
      Priority=1
```

```
AutoPriority=false
```

Tip: *Naturally, you could repeat this process to identify additional inventory beacons to make a set that the inventory device can choose between. Keep in mind that:*

- *`AutoPriority` must be set to `false`, to prevent the selector algorithm making any of your approved inventory beacons invalid for upload/download*
- *`Priority` must then be set manually to the order in which you want the target inventory device to attempt communications with the inventory beacons in your set.*

3. Use your preferred administrative tool to deploy registry settings to target Windows devices, or to deploy a customized `config.ini` to UNIX-like devices.

After installation of the custom settings in the platform-appropriate manner, the affected inventory devices access only the inventory beacon(s) you specified in the special settings package. Do remember that any network changes that affects devices and their inventory beacons must now be manually managed. If you wish to revert to standard, self-managing behavior for some or all inventory devices, simply remove their custom values for `UploadSettings` and `DownloadSettings`, and restore the default value for the `SelectorAlgorithm`.

Supplied Algorithms

FlexNet Manager Suite includes the following algorithms that inventory devices may use for assigning priorities to inventory beacons:

- `MgsBandwidth`: Priorities are based on end-to-end bandwidth availability to the server
- `MgsDomainMatch`: Priorities are determined by closest match in domain name

Tip: *On UNIX-like platforms, using this algorithm requires that you have first set the `ComputerDomain` preference to a valid domain name. This is used as the domain of the inventory device for comparison with the domains of inventory beacons.*

- `MgsIPMatch`: Priorities are determined by closest IP address match
- `MgsNameMatch`: Matches prefixes in computer names
- `MgsPing`: Priorities are determined by fastest ping response time
- `MgsRandom`: Random priorities are assigned
- `MgsSubnetMatch`: Moves all servers in the current subnet to the front of the priority list, but retaining the relative order of existing priorities.

The preceding algorithms may be used on both Windows and UNIX-like platforms. In addition, the following algorithms are available only for inventory devices running Microsoft Windows:

- `MgsADSiteMatch`: Moves all servers in the current managed device's site to the front of the priority list
- `MgsDHCP`: Priorities are based on lists of servers specified in DHCP

- `MgsServersFromAD`: Priorities are determined according to lists of servers specified in Active Directory.

Tip: *When unable to differentiate between locations, the supplied algorithms assign priorities according to the order in which locations are listed in the registry settings.*

How you use these algorithms depends on the structure of your hierarchy of inventory beacons, and how you want to spread the load of file uploads and downloads across that hierarchy.

For example, *random* priorities may be useful if you want to ensure a good load balance across inventory beacons, while *domain matching* may tend to favor particular inventory beacons.

You can choose a combination of fixed and assigned dynamic priorities, where you determine the priorities of some servers, but allow the priorities of other servers to be assigned by the supplied algorithms. For example, there may be a number of inventory beacons to which you want to specifically assign low priorities, but let the inventory device randomly assign priorities amongst the inventory beacons you want to use more often. In this scenario, for each location that requires a fixed priority, you would set the `AutoPriority` preference to `false` and manually assign a value for the `Priority` preference.

For mobile users, whose upload and download speeds can be severely affected by the decision about which inventory beacon to use, the *IP matching* algorithm is most often favored. When a mobile user connects to the network and is allocated an IP address, the inventory device uses the inventory beacon with the closest match to that IP address, matching components of the IP address from left to right. For example, when a mobile user who travels to different geographic regions is in Germany, the IP matching algorithm assigns higher priority to the company's European inventory beacons. When the same mobile user travels to New York, the company's US inventory beacons get highest priority.

Algorithms like *site matching* and *name matching* can be useful in large enterprises. Among other effects, you can use them to limit the range of fastest response servers, and thereby control network impacts.

You can use more than one algorithm. When you do this, the inventory device assigns priorities according to the first algorithm, then reassigns priorities according to subsequent ones. This can be used to refine priorities, or to ensure a shared load across inventory beacons, as the following example shows:

```
SelectorAlgorithm="MgsDomainMatch;MgsRandom(3)"
```

This means that the inventory device should sort all inventory beacons using a domain match, then randomize the top three inventory beacons. As another example, using a combination of IP matching and random algorithms would help balance the load between inventory beacons with similar IP addresses.

An empty string for `SelectorAlgorithm` means that no algorithm is used to determine the download or upload location. In this case, the most recent priority values assigned to locations are used.

By combining algorithms, you can also limit the number of inventory beacons that other algorithms will test, since they will all ignore any inventory beacon with a priority set to `invalid` (or to any string that does not represent an integer). For example, a combination such as:

```
MgsADSiteMatch(, true);MgsSubnetMatch(, true);MgsPing(3)
```

will prioritize the fastest three inventory beacons within the target inventory device's site and subnet. The important thing to notice is that *only* devices within the Active Directory site are checked for subnet matching; and *only* the devices in the same subnet as the inventory device are pinged to determine response time. If the order were different, and `MgsPing` were first, the inventory device would have tried pinging *every* inventory beacon in the failover list.

Remember that the algorithms only assign values to locations that have the `AutoPriority` registry key set to `true`.

Details of each of the supplied algorithms follow.

MgsADSiteMatch: Match to Active Directory Site

Prerequisites

- The target inventory device is running Windows 2000 or later
- The target inventory device must be joined to an Active Directory domain
- **Active Directory Sites and Services** must be correctly configured.

This algorithm is particularly useful when the failover list includes a large number of inventory beacons (perhaps all the inventory beacons in your enterprise), and you want to restrict transfers to those within the same site that contains the inventory device. `MgsADSiteMatch` takes the following steps:

1. Identifies the Active Directory site in which this inventory device is a member.
2. Collects the list of selected inventory beacons from the preference settings (normally in the machine hive of the registry).
   - If the algorithm is in use by the launcher component of the FlexNet inventory agent (for example, collecting updated device policy), it collects the inventory beacon list from the preference `DownloadSettings`
   - If the algorithm is in use by the upload component, it collects the inventory beacon list from the preference `UploadSettings`.
3. For each inventory beacon in the list, it looks up the Active Directory site in which the inventory beacon is a member. You can restrict the domain controller used for these inventory beacon site queries using the `localSiteDCOnly` parameter (see below).
4. If the inventory beacon's site is the same as the site for the target inventory device, the algorithm assigns a high priority. Where there are multiple such inventory beacons, their relative priority is left unchanged.
5. If the inventory beacon is in a different site than the inventory device, the behavior depends on the `discardForeign` parameter (see below).

Syntax:
`MgsADSiteMatch` (int *limit*, boolean *discardForeign*, boolean *localSiteDCOnly*)

where:

- *`limit`* is an optional integer setting the maximum number of inventory beacons to which priorities will be assigned by this algorithm.
- *`discardForeign`* is an optional boolean represented by the case-insensitive strings `true` or `false`. The default is `false`.
  - If `false` or omitted, the algorithm assigns a low priority to inventory beacons that are not in the same Active Directory site as the target inventory device. Where there are multiple such inventory beacons, their relative priority is maintained.

- If `true`, the algorithm sets the priorities of any inventory beacons that are not in the same Active Directory site as the target inventory device to the string literal `invalid`. The launcher component and the upload component will not use such inventory beacons for transfers.

- `localSiteDCOnly` is an optional boolean represented by the case-insensitive strings `true` or `false`. The default is `false`.
  - If `false` or omitted, the algorithm does not require the domain controller used for inventory beacon site queries to be in the same Active Directory site as the target inventory device.
  - If true, the algorithm restricts the inventory beacon site queries to a domain controller that is in the same Active Directory site as the target inventory device. If there is no domain controller in the same site as the inventory device, the algorithm treats all inventory beacons as being in a different Active Directory site than the target inventory device.

Example: Example of MgsADSiteMatch algorithm results

For target inventory device in Active Directory site: boston

Location	AD site of inventory beacon	AutoPriority	discard-Foreign	Incoming priority	Resulting priority
A	melbourne	true	false true	1	2 invalid
B	boston	true	false true	3	1 1
C	boston	false	Don't care	4	4
D	frankfurt	true	false true	2	3 invalid

In the above example, the algorithm determines priorities in the following way:

- Location C has `AutoPriority` set to `false`, so that it is excluded from the calculation process and its priority value of 4 is carried through unchanged.
- When the algorithm parameter `discardForeign` is `false` or missing, Locations A and D will be given lower priorities (with their relative order maintained). However, when `discardForeign` is true, both Locations A and D will be marked invalid because they are not in the same site as the target inventory device.

MgsBandwidth: Bandwidth Priorities

This algorithm prioritizes inventory beacons based on the end-to-end bandwidth available to each inventory beacon. It uses an average of ping requests where packets of different sizes are sent as part of the calculation. Unlike `MgsPing`, there is no parallelism in querying servers, so that this algorithm should only be used in scenarios that do not require parallel pinging.

`MgsBandwidth` estimates the total bandwidth available between the local inventory device and each inventory beacon, and *not* the amount of currently unused bandwidth. The estimate is more accurate when there is less traffic on the network.

Tip: *The bandwidth calculation used is very similar to the one described in http://www.microsoft.com/resources/documentation/windows/2000/ server/reskit/en-us/distrib/dsec_pol_chzb.asp.*

Syntax:
`MgsBandwidth` (int *limit*)

where:

- *`Limit`* is an optional integer setting the maximum number of inventory beacons to which priorities will be assigned by this algorithm.

Example: Example of MgsBandwidth algorithm results

Location	Bandwidth	Incoming priority	Resulting priority
A	LAN (100Mbps)	blank	1
B	LAN (10Mbps)	blank	2
C	WAN (56K Modem)	blank	4
D	WAN (ADSL)	blank	3

MgsDHCP: Retrieve location list from DHCP server options

Prerequisites

- This algorithm relies on you having specified a known option number with a prioritized list of inventory beacons on each DHCP scope in your enterprise.
- Values may be distinct in each scope.
- The inventory beacons listed in the value should be those that you wish inventory devices within the DHCP scope to access. For instructions on how to configure the DHCP server option, see the next page.

This algorithm prioritizes inventory beacons according to lists of inventory beacons that you have specified in a DHCP property. Whenever the installation or upload components need to prioritize distribution or reporting locations, NetSelector sends a DHCP broadcast requesting the value of the DHCP property corresponding to the option number specified as a parameter to the algorithm. The value returns a list of one or more inventory beacons in priority order.

Syntax:
`MgsBandwidth` (int *limit*, boolean *discardForeign*, int *option*)

where:

- *limit* is an optional integer setting the maximum number of inventory beacons to which priorities will be assigned by this algorithm. If limit is not specified (empty), all servers listed in the DHCP server option will be prioritized.
- *discardForeign* is an optional boolean represented by the case-insensitive strings `true` or `false`. The default is `false`.
  - If `false` or omitted, the algorithm assigns a lower priority to inventory beacons that are not prioritized by the algorithm.
  - If `true`, the algorithm sets the priorities of any inventory beacons that are not prioritized by the algorithm to the string literal `invalid`. The launcher component and the upload component will not use such inventory beacons for transfers.
- *option* is an optional integer identifying the option configured on the DHCP server that contains the failover inventory beacon information. Valid values are 0 to 255. The default is 123.

For example:

```
MgsDHCP(,true,123)
```

Example: Example of MgsDHCP algorithm results

For DHCP server option set to `ds-prs-01.tmnis.org,ds-prs-02.tmnis.org`.

Location	AutoPriority	discard-Foreign	Incoming priority	Resulting priority
ds-cls-01	true	false true	1	2 invalid
ds-prs-01	false	Don't care	4	4
ds-cls-02	true	false true	2	3 invalid
ds-prs-02	true	Don't care	3	1

In the above example:

- The fact that `AutoPriority` has been set to `false` for `ds-prs-01` prevents it from being given the highest priority, despite its pre-eminent position in the DHCP option listing. It must preserve its incoming priority value.
- The inventory beacons `ds-cls-01` and `ds-cls-02` are not prioritized in the DHCP option. Therefore, depending on the value of `AutoPriority`, they are either sent to the end of the priority queue (maintaining the relative ordering they had on entry), or excluded.

Configuring the DHCP Option

The DHCP option must be a string type and have an option number matching the one supplied to the `MgsDHCP` algorithm. Typically, you need to configure this on each DHCP scope within your enterprise on which there are target

inventory devices.

The syntax of the value is:

```
serverlist[servername | random(servername[,...n]),...n]
```

Some examples are:

- `srv1,srv2,srv3`: Prioritizes srv1 first, followed by srv2 and srv3
- `random(srv1,srv2,srv3)`: Prioritizes the three servers in random order
- `random(srv1,srv2,srv3),srv4`: Prioritizes srv2, srv1, srv3 in random order, followed by srv4
- `random(srv1,srv2,srv3),random(srv4,srv5,srv6)`: Prioritizes srv1, srv2, and srv3 in random order, then srv4, srv5, and srv6 in random order
- `srv0,random(srv2,srv2,srv3),random(srv4,srv5,srv6)`: Prioritizes srv0 first, followed by srv1, srv2, and srv3 in random order, then srv4, srv5, and srv6 in random order
- `srv0,random(srv1,srv2,srv3),srv4`: Prioritizes srv0 first, followed by srv1, srv2, and srv3 in random order, followed lastly by srv4.

The details about configuring a DHCP option naturally depend on the DHCP server that you are using. These instructions describe how to configure the DHCP option on a Microsoft DHCP Server.

To configure the DHCP server (Microsoft example):

1. Create a custom DHCP scope option:
   a. Start the DHCP MMC snap-in.
   b. Right-click the server name and select **Set Predefined Options...**

      The **Predefined Options and Values** dialog is displayed.
   c. Click **Add...**
   d. In the **Name** field, enter a descriptive name for the option.
   e. In the **Code** field, enter an option number such as `123`. This is the option parameter that is passed to `MgsDHCP`.
   f. Set the **Data type** to `string`.
   g. Click **OK**.
2. Enable and configure the DHCP option:
   a. Start the DHCP MMC snap-in.
   b. Right-click ***<Scope Name>*** **Scope Options** and select **Configure Options**.
   c. Select the option and enter the server list in the **String** field.
   d. Click **OK**.
3. Repeat the previous step for every DHCP scope.

MgsDomainMatch: Match to Domain Name

This algorithm prioritizes inventory beacons based on their domain name. The name closest to that of the target inventory device is given the highest priority.

`MgsDomainMatch` checks each domain component, from right to left.

Tip: *On UNIX-like platforms, using this algorithm requires that you have first set the* `ComputerDomain` *preference to a valid domain name. This is used as the domain of the inventory device for comparison with the domains of inventory beacons.*

Syntax:
`MgsDomainMatch` (int *`limit`*)

where:

- *`limit`* is an optional integer setting the maximum number of inventory beacons to which priorities will be assigned by this algorithm.

Example: Example of MgsDomainMatch algorithm results
For a target inventory device in the domain `abc.com.au`.

Tip: *Because multiple domains may be assigned the same priority, the NetSelector indexes priorities according to the order in which domains are listed in the registry (in the following example, shown in the* ***Location*** *column).*

Location	Domain of inventory beacon	AutoPriority	Incoming priority	Resulting priority	Normalized priority
A	abc.com.au	true	blank	6	1
B	abb.com.au	true	blank	42	2
C	abc.com.de	true	blank	83	3
D	abb.com.de	true	blank	104	4
E	abb.com.de	false	5	5	5

In the above example:

- Location A matches the domain name of the target inventory device, so it is assigned a very high priority (for example, 5). This priority is then indexed by the order of locations in the registry. It is first, so the priority assigned is 5+1 (6).
- Location B matches the third and second domain components, but not the first. It is given a medium priority (for example, 40). After indexing, the priority is 40 + 2 (42).
- Location C does not match the third domain component, so it is given a lower priority (for example, 80).

After indexing, the priority is 80 + 3 (83).

- Location D only matches the middle domain component and is given a very low priority (for example, 100). After indexing, the priority is 100 + 4 (104).
- Location E has a fixed priority, so priority 5 is unchanged by the algorithm.
- The NetSelector normalizes the priorities of locations A to D to fit within the range specified by the server selection settings (described in Prioritizing Inventory Beacons). In this example, the range is 1-5.

MgsIPMatch: Match to IP Address

This algorithm prioritizes inventory beacons based on similarities in the IP address of the target inventory device and each inventory beacon. Address components are converted to binary numbers and compared, left to right. Priority is assigned according to the longest common (matching) bit in the binary number.

By comparing the IP address of the inventory device against each inventory beacon, any inventory beacons within a subnet will be given higher priority because the network address portion of the IP will be the same. If two inventory beacons are within the same subnet, then the value of the local host ID determines the priorities of the two inventory beacons.

Syntax:
`MgsIPMatch` (int *limit*)

where:

- *limit* is an optional integer setting the maximum number of inventory beacons to which priorities will be assigned by this algorithm.

Example: Example of MgsIPMatch algorithm results
For a target inventory device with IP address `123.3.45.44`.

Tip: *This algorithm only supports the IPv4 address family.*

Location	IP address of inventory beacon	AutoPriority	Incoming priority	Resulting priority	Normalized priority
A	123.3.45.56	true	blank	21	2
B	123.3.44.46	true	blank	42	3
C	123.5.45.56	true	blank	63	4
D	123.3.45.32	true	blank	14	1
E	123.3.45.42	false	5	5	5

In the above example:

- Location A matches the first three domain components, so there is little binary difference from the managed device's IP address. The algorithm assigns a high priority (for example 20). It is then indexed by the order of location entries in the registry. Location A is first in the list, so the priority is recalculated as 20+1 (21).
- Location B matches on the first and second address components and has a greater binary difference than location A. It is given a medium high priority (for example, 40). This is indexed and the priority is recalculated as 40 + 2 (42).
- Location C matches the managed device on the first domain component and the third component, but not on the last address component. It has a high binary difference and is given a low priority (for example, 60). This is indexed and the priority is recalculated as 60 + 3 (63).
- Location D is similar to location A because it doesn't match the last domain component of the managed device, but it does match on the first three domain components. It actually has a smaller binary difference, and is assigned a very high priority (for example, 10). It is then indexed by the order of location entries in the registry; it is fourth in the list, so the priority is recalculated as 10 + 4 (14).
- Location E has a fixed priority, so priority 5 is unchanged by the algorithm.
- The NetSelector normalizes the priorities of locations A to D to fit within the range specified by the server selection settings (described in Prioritizing Inventory Beacons). In this example, the range is 1-5.

MgsNameMatch: Match Prefixes of Computer Names

This algorithm compares the leading characters of the host name of each inventory beacon with the first characters of the name of the target inventory device on which the algorithm is running. Those inventory beacons whose names match according to this check are prioritized above those whose names do not match, while retaining the relative order of existing priorities.

The name of the inventory device on which the algorithm is running is determined by the `MachineName` preference (see MachineName).

`MgsNameMatch` may be useful in situations where computers' names have a prefix determined according to their location or site. It is common to use this algorithm in conjunction with another algorithm, such as:

```
MgsNameMatch(5,,true);MgsRandom
```

This combination selects all inventory beacons that have at least five matching characters in their names, and then randomizes access (to achieve load balancing).

Syntax:
`MgsNameMatch` (int *matchLength*, int *limit*, boolean *discardForeign*)

where:

- *`matchLength`* is the number of characters to compare in the target inventory device and inventory beacon host

names.

- *limit* is an optional integer setting the maximum number of inventory beacons to which priorities will be assigned by this algorithm.
- *discardForeign* is an optional boolean represented by the case-insensitive strings `true` or `false`. The default is `false`.
  - If `false` or omitted, the algorithm assigns a lower priority to inventory beacons that are not matched by the algorithm than to all those that are matched.
  - If `true`, the algorithm sets the priorities of any unmatched inventory beacons to the string literal `invalid`. The launcher component and the upload component will not use such inventory beacons for transfers.

Example: Example of MgsNameMatch algorithm results

For a target inventory device with `MachineName Bost0014`.

Host name of inventory beacon	match-Length	discardForeign	Incoming priority	Resulting priority
BostonS01	5	true	5	invalid
BostonS02	4	don't care	3	2
ChicagS03	4	false	2	4
BostonS06	4	don't care	1	1
BostonS04	4	don't care	4	3

In the above example:

- In the first row, the *matchLength* is set so high that the name matching fails on the fifth character. Since *discardForeign* is `true`, the priority after a failure must be set to `invalid`, and this inventory beacon will be ignored. (In the remaining rows, the *matchLength* is corrected to 4.)
- All the remaining Boston inventory beacons pass the matching test, so their priorities are maintained in the same relative order.
- The Chicago inventory beacon fails the match. Since *discardForeign* is `false`, it remains in the list, but its priority is set lower than all matching inventory beacons.

MgsPing: Server with the Fastest Response

This algorithm pings each inventory beacon three times with a timeout of three seconds. An unlimited number of inventory beacons can be prioritized, but a maximum of 50 are 'pinged' in parallel at the same time. The general rule is that for 50 actively responding servers, the prioritization will be complete in fewer than five seconds.

The inventory beacon with the shortest response time is assigned the highest priority, and so on down the list ordered by response times.

Warning: *Carefully consider the multiplier effect of using this algorithm on a large number of target inventory devices addressing a large number of inventory beacons. It could have a negative impact on network performance, particularly at tightly-scheduled policy update times. The* `maxHops` *parameter is useful for limiting the scope of pings.*
Where pinging is desirable, consider combining it with another algorithm to limit the set of servers tested.

Syntax:
`MgsPing` (int *limit*, boolean *discardForeign*, int *maxHops*)

where:

- *`limit`* is an optional integer setting the maximum number of inventory beacons to which priorities will be assigned by this algorithm.
- *`discardForeign`* is an optional boolean represented by the case-insensitive strings `true` or `false`. The default is `false`.
  - If `false` or omitted, the algorithm assigns a lower priority to inventory beacons that are not matched by the algorithm than to all those that are matched.
  - If `true`, the algorithm sets the priorities of any unmatched inventory beacons to the string literal `invalid`. The launcher component and the upload component will not use such inventory beacons for transfers.
- *`maxHops`* is an optional integer setting the maximum number of network segments that the ping will traverse. A value of 1 means that there must be no routers between the two computers. Valid values are 1 to 255. The default is 127.

Example: Example of MgsPing algorithm results

Location	Avg round-trip time	Incoming priority	Resulting priority
A	100 ms	blank	2
B	50 ms	blank	1
C	200 ms	blank	3
D	400 ms	blank	4

MgsRandom: Random Priorities

This algorithm assigns random priorities to the download/upload locations on inventory beacons, within the range specified by the `HighestPriority` and `LowestPriority` registry keys (see HighestPriority and LowestPriority). This ensures that all target inventory devices referencing the same failover list of locations do not use the same location for download and/or upload, spreading the load between inventory beacons.

Syntax:
`MgsRandom` (int *limit*)

where:

- *Limit* is an optional integer setting the maximum number of inventory beacons to which priorities will be assigned by this algorithm.

Example: Example of MgsRandom algorithm results
For:

- Domain of target inventory device: abc.com.au
- IP address of target inventory device: 123.3.45.44

In the table below, the domain and IP address are those of the inventory beacon under consideration.

Location	Domain	IP address	Auto Priority	Incoming priority	Resulting priority
A	abc.com.au	123.3.45.56	true	blank	3
B	abc.com.de	123.3.44.46	true	blank	1
C	abb.com.au	123.5.45.56	true	blank	2
D	abbb.com.de	123.3.45.32	true	blank	4
E	abc.com.au	123.3.45.42	false	5	5

In the above example, the algorithm determines priorities in the following way:

- Location E has `AutoPriority` set to `false`, so that it is excluded from the calculation process and its priority value of 5 is carried through unchanged.
- Priorities 1-4 are randomly assigned to the remaining locations.

MgsServersFromAD: Retrieve List from AD

This algorithm prioritizes according to lists of inventory beacons specified in Active Directory. Consequently, it is not useful for inventory devices that cannot run the Active Directory client:

- Non-Windows devices that are not known to Active Directory
- Any target inventory devices running legacy Windows operating systems that are without the Active Directory client.

Note: *To configure this algorithm, you need to use* ***ADSI Edit*** *(`adsiedit.msc`), a GUI tool that acts as a low-level editor for Active Directory.*

Syntax:
`MgsServersFromAD` (int *limit*, boolean *discardForeign*, string *dnPrefix*)

where:

- *limit* is an optional integer setting the maximum number of inventory beacons to which priorities will be assigned by this algorithm.
- *discardForeign* is an optional boolean represented by the case-insensitive strings `true` or `false`. The default is `false`.
  - If `false` or omitted, the algorithm assigns a low priority to inventory beacons that are not in the same Active Directory site as the target inventory device. Where there are multiple such inventory beacons, their relative priority is maintained.
  - If `true`, the algorithm sets the priorities of any inventory beacons that are not in the same Active Directory site as the target inventory device to the string literal `invalid`. The launcher component and the upload component will not use such inventory beacons for transfers.
- *dnPrefix* is the prefix (quoted with double quotation marks) to add to a computer's distinguished name (DN), its subnet DN and its site DN in order to find an object in Active Directory with failover information. Failover information is obtained from the `Description` attribute in the first such object found in AD. Defaults to `"CN=ManageSoft"` if not specified.
  As usual for distinguished names, any of the following special characters must be escaped with a backslash (`\`) character wherever they appear in distinguished name components (see following example):

  ```
  , = + < > # ;
  ```

  Additionally, any double quote (`"`) characters in `dnPrefix` must be similarly escaped.

Example:

```
MgsServersFromAD(,true,"CN=MGS Servers\, For Failover")
```

Including this example specification in the `MgsServersFromAD` preference has the following effects:

- Gives high priority to all inventory beacons specified in the `Description` attribute of whichever of the following objects is found first within Active Directory:
  - `CN=MGS Servers\, For Failover,CN=<computer name>,<computer's OU/ container DN>`
  - `CN=MGS Servers\, For Failover,CN=<subnet>,CN=Subnets,CN=Sites, CN=Configuration,<domain DN>`
  - `CN=MGS Servers\, For Failover,CN=<site name>,CN=Sites, CN=Configuration,<domain DN>`
- Discards any inventory beacons not specified in Active Directory from the failover list.

When the target inventory device uses this algorithm, it looks up each of these three locations (computer name, subnet, and site name) in Active Directory based on the configuration of that inventory device. Each inventory device calculates its own set of these three Active Directory queries.

Therefore, for a particular target inventory device:

- Inventory device `MachineName`: `mypc`
- Inventory device OU: `Desktops`
- Inventory device domain: `abc.com`

- Inventory device subnet: `172.16.34.0`
- Inventory device site: `melbourne`

including the above example specification in the `MgsServersFromAD` preference causes the inventory device to check for Active Directory objects in the following order:

1. `CN=MGS Servers\, For Failover,CN=mypc,OU=Desktops, DC=abc,DC=com`
2. `CN=MGS Servers\, For Failover,CN=172.16.34.0,CN=Subnets,CN=Sites, CN=Configuration,DC=abc,DC=com`
3. `CN=MGS Servers\, For Failover,CN=melbourne,CN=Sites, CN=Configuration,DC=abc,DC=com`

If one of these objects is found, the inventory device checks the `Description` attribute of this object and extracts the inventory beacon information from this list. The failover information must be a comma-separated list of inventory beacon host names, using the following syntax:

```
serverlist[,...n]
```

where `serverlist` = `[servername | random(servername[,...n])]`.

Some examples are:

- `srv1,srv2,srv3` — Prioritizes srv1 first, followed by srv2 and srv3
- `random(srv1,srv2,srv3)` — Prioritizes srv1, srv2, and srv3 in random order
- `random(srv1,srv2,srv3),srv4` — Prioritizes srv1, srv2, srv3 in random order, followed by srv4
- `random(srv1,srv2,srv3),random(srv4,srv5,srv6)` — Prioritizes srv1, srv2, andsrv3 in random order, then srv4, srv5, and srv6 in random order
- `srv0,random(srv2,srv2,srv3),random(srv4,srv5,srv6)` — Prioritizes srv0 first, followed by srv1, srv2, and srv3 in random order, then srv4, srv5, and srv6 in random order
- `srv0,random(srv1,srv2,srv3),srv4` — Prioritizes srv0 first, followed by srv1, srv2, and srv3 in random order, followed lastly by srv4.

Example: Example of MgsServersFromAD algorithm results

For `Description` attribute set to `ds-prs-01.tmnis.org,ds-prs-02.tmnis.org`:

Beacon	Auto Priority	discard-Foreign	Incoming priority	Resulting priority
`ds-cls-01`	true	false true	1	2 invalid
`ds-prs-01`	false	false true	4	4 4
`ds-cls-02`	true	false true	2	3 invalid

Beacon	Auto Priority	discard-Foreign	Incoming priority	Resulting priority
ds-prs-02	true	false true	3	1 1

In the example shown, the fact that AutoPriority has been set to false for ds-prs-01 prevents it from being given the highest priority, despite its preeminent position in the Active Directory listing.

Taking care not to orphan inventory devices

When you set the discardForeign flag to true, any inventory beacons not found in Active Directory are discarded. There is a possibility that inventory devices may become orphaned from all inventory beacons if the inventory device's list of download locations does not contain any of the servers listed in Active Directory. If this occurs, the inventory device will not attempt to download any packages, including any updated failover settings packages.

If a value is not specified for discardForeign, it defaults to false. In this case, when a target inventory device cannot use any inventory beacon listed in Active Directory, it will still be able to use other inventory beacons identified through other algorithms.

MgsSubnetMatch: Match to Subnet

This algorithm scans through all inventory beacons in the subnet containing the target inventory device and moves them to the front of the priority list, while retaining the relative order of existing priorities.

This is particularly useful if combining with other algorithms, such as

```
MgsPing;MgsSubnetMatch
```

This combination provides a resultant set of priorities based on the fastest responding servers within the current subnet, followed by the fastest responding servers outside the subnet.

Syntax:
MgsSubnetMatch (int *limit*, boolean *discardForeign*)

where:

- *limit* is an optional integer setting the maximum number of inventory beacons to which priorities will be assigned by this algorithm.
- *discardForeign* is an optional boolean represented by the case-insensitive strings true or false. The default is false.
  - If false or omitted, the algorithm assigns a lower priority to inventory beacons that are not matched by the algorithm than to all those that are matched.
  - If true, the algorithm sets the priorities of any unmatched inventory beacons to the string literal invalid. The launcher component and the upload component will not use such inventory beacons for transfers.

Example: Example of MgsSubnetMatch algorithm results

For a target inventory device with:

- IP address: `172.16.34.53`
- Subnet mask: `255.255.248.0`
- Subnet: `172.16.34.0`

In the following table, the bandwidth and IP address are for each inventory beacon under consideration.

Location	Bandwidth	IP address	Incoming priority	Resulting priority
Location A	LAN (100Mbps)	172.16.45.40	1	3
Location B	LAN (10Mbps)	169.15.13.12	2	4
Location C	WAN (56K modem)	172.16.34.40	4	2
Location D	WAN (ADSL)	172.16.34.60	3	1

In the above example:

- Location C and D are in the same subnet as the client, and therefore have higher priority than A and B.

9

Command Lines

To assist with the preparation of custom solutions, this section summarizes the command lines for the various code entities that together function as the FlexNet inventory agent. The mapping of descriptive titles to executable names is:

- FlexNet inventory agent includes many components, but the inventory-gathering component is `ndtrack`
- Installation component is `ndlaunch`
- Policy component is `mgspolicy`
- Schedule component is `ndschedag`
- Upload component (or uploader) is `ndupload`.

The application usage component does not have any command line available (on Windows, it runs as a plug-in, and on UNIX-like systems, it runs as a service or daemon). However, there are still a few preferences relevant to the application usage component. Since there is no command line, these can only be adjusted by directly editing the registry (or the `config.ini` file on UNIX-like platforms). Relevant items are listed in the chapter Preferences.

mgspolicy Command Line

The policy component is responsible for checking and applying policy on the managed device.

Because server-side policy merging is run on the inventory beacons, the policy component simply invokes the installation component. The installation component retrieves the resulting policy files from the appropriate inventory beacon, and installs the required packages. For command line details for the installation component, see ndlaunch Command Line.

During adoption, the policy component attempts to collect policy and its operating schedule from the bootstrapping inventory beacon (which has collected these from the central application server). If for any reason this collection does not succeed, then until the FlexNet inventory agent receives its operating schedule, the policy component runs:

- On Windows platforms, each time the managed device reboots
- On UNIX-like platforms, once each day at a random time between 0800 and 2300 (local time on the managed device).

Once the operating schedule is received from the bootstrapping inventory beacon, this behavior is discontinued, and the policy component is triggered regularly to check for changes, and apply the policy when amended. The policy

component uses preference settings on the managed device to determine the appropriate command line parameters.

Synopses

Syntax:
mgspolicy [*options* ...]

Options:

Syntax:
-o *tag* = *value*
-s *source*
-t User | **Machine**

-o	*tag*=*value*	Each instance over-rides the specified preference for the policy component. Each parameter set at the command line must be accompanied by its own `-o` flag. Do not repeat any individual tag within the command line. Possible tags are listed below. ***Note:*** *In addition to the tags listed below, you may also set any preferences for the inventory component (`ndtrack`) on the command line. When the policy component launches `ndtrack`, these preferences will be passed to the `ndtrack` command line. When you set policy agent logging preferences on the command line, these are also passed to `ndtrack`, and the log file becomes a single combined file for both `mgspolicy` and `ndtrack` activities. For details, see ndtrack Command Line.*
-s	*source*	Identifies the source location of server-side merged policy (`.npl`) files on the inventory beacon. If you do not specify a source, the policy agent uses the last known location (set in the `PolicyServerURL` preference). Example: `-s "http://beacon.mydomain.com/ManageSoftDL/Policies/Merged/Machine/deviceHostname.npl"`
-t	`User`|`Machine`	Identifies the type of policy to be merged. This can be user or machine policy. ***Note:*** *User-based policy is now deprecated.*

Options

Possible tags for use in a command line with the `-o` options are shown below.

Note: *Most preferences for the policy agent are not passed through to the installation agent when it is called. Either list the preferences in the `Common` registry key, or use separate preferences for the installation agent.*

- LogFile (policy component)

- LogFileOld (policy component)
- LogFileSize (policy component)
- LogLevel (policy component)
- LogModules (policy component)
- MachinePolicy
- MachinePolicyDirectory
- MachinePolicyPackageDirectory
- PolicyServerURL.

In addition, the following preferences set in the registry on the managed device influence the behavior of the policy agent:

- DefaultSchedulePath
- InstallDefaultSchedule
- LauncherCommandLine
- StrictInstall.

The following preferences available in earlier releases are now deprecated, and should not be used:

- `AllowedPkgSubtypes`
- `AutoDetectDC`
- `MinimumDCSpeed`
- `PolicySource`
- `UserPolicy`
- `UserPolicyDirectory`
- `UserPolicyPackageDirectory`.

ndlaunch Command Line

The installation component on managed devices is now limited to installing various packages required by the FlexNet inventory agent, including updating the FlexNet inventory agent itself.

Synopses

Syntax:

`ndlaunch` [**`-f`** *preferenceFile* | **`-o`** *tag* = *value*] [*catalogFile*]

`ndlaunch -a` *PackageName* [**`-o`** *tag* = *value* ...]

`ndlaunch -d` *PackageName*

ndlaunch -r *urlOfCatalogFile* [**-o** *tag* = *value* ...]

-a	*PackageName*	Updates the named package from the last location from which it was downloaded and installed. If no update is necessary or available, runs the currently installed version (if execution is required). This parameter is equivalent to using both `-p` and `-r`. This is the mechanism by which the FlexNet inventory agent checks for any changes to its policy. Examples: `%BASE\ndlaunch –a mypolicy` ***Tip:*** *Use the %BASE path substitution variable to include the full path to the FlexNet inventory agent installation.*
	catalogFile	The full pathname to a locally available FlexNet catalog. If specified, the *catalogFile* filename must appear last on the command line. FlexNet catalogs have the file extension `.osd`.
-d	*PackageName*	Delete the application identified by the package name. You can identify the package name of any installed application by looking in the installation component's application cache; each installed application has its own folder (named for the associated package) in this location. ***Warning:*** *Do not use this parameter to delete standard packages distributed by FlexNet Manager Suite through the inventory beacons.*

-f	*preferenceFile*	Advanced parameter, recommended for use by experienced administrators only. Sets the value of a symbol used in a catalog by passing it the contents of a preference file using the installation component's command line. The installation component reads the list of symbol values specified in *preferenceFile*, and substitutes them for the symbols used in the catalog.
		preferenceFile uses the standard `.ini` file format. For example:
		`[Symbols]`
		`symbolName="value"`
		`symbolName2="value2"`
		`symbolName3="value3"`
		where:
		• `symbolName` is the name of a symbol for which you wish to define a value
		• `value` is the value assigned to the symbol.

-o	*tag=value*	Each instance over-rides the specified preference for the installation component. Each parameter set at the command line must be accompanied by its own `-o` flag. Do not repeat any individual tag within the command line. Possible tags are listed below. If your command line will exceed the length limit of your operating system's command line, use the `-f` parameter instead.
-r	*urlOfCatalogFile*	Check to see if an updated catalog file exists at the specified URL. If so, download and install it. Example: `%BASE\ndlaunch -r http://www.mysite.org/myapp.osd`

Options

Possible tags for use with the `-o` options are:

- AddClientCertificateAndKey (UNIX-like platforms only)
- AllowedPkgTypes
- CheckCertificateRevocation
- CheckServerCertificate
- DownloadSettings
- http_proxy
- InstallProfile
- LogFile (installation component)
- LogFileOld (installation component)
- LogFileSize (installation component)
- NetworkHighSpeed
- NetworkHighUsage
- NetworkHighUsageLowerLimit
- NetworkHighUsageUpperLimit
- NetworkLowUsage
- NetworkLowUsageLowerLimit
- NetworkLowUsageUpperLimit
- NetworkMaxRate
- NetworkMinSpeed
- NetworkSense
- NetworkSpeed

- no_proxy
- PreferIPVersion
- PrioritizeRevocationChecks (UNIX-like platforms only)
- SSLCACertificateFile (UNIX-like platforms only)
- SSLCACertificatePath (UNIX-like platforms only)
- SSLClientCertificateFile (UNIX-like platforms only)
- SSLClientPrivateKeyFile (UNIX-like platforms only)
- SSLCRLCacheLifetime (UNIX-like platforms only)
- SSLCRLPath (UNIX-like platforms only)
- SSLDirectory (UNIX-like platforms only)
- SSLOCSPCacheLifetime (UNIX-like platforms only)
- SSLOCSPPath (UNIX-like platforms only)
- StrictInstall
- UserInteractionLevel (installation component)

In previous releases, `ndlaunch` had greater functionality and many more preferences. The following are now deprecated (meaning that their use is discouraged, that their ongoing functionality is not guaranteed, and that at an unspecified future time they may be entirely removed from the `ndlaunch` component):

`AddRemove`	`AlertExecute`	`AllowByteLevel`
`AllowedPkgSubtypes`	`AllowRebootIfLocked`	`AllowRebootIfServer`
`AllowTimeoutIfLocked`	`AlwaysDisplayReboot`	`ApplyPolicy`
`AskAboutDependencies`	`AutoAlertExecute`	`AutoDetectDC`
`AutoPromptOn InstallCompletion`	`AutoPromptOn UnInstallCompletion`	`AutoRedundancy`
`BrandARP`	`CacheDir`	`CacheDirectory`
`CachedVersion`	`CheckCatalogDigest`	`CheckFileDigest`
`CheckRegistry`	`CmdLineOverrides`	`CompressMSA`
`ConfirmSharedFileRemoval`	`ConnectionAttempts`	`DetectApplicationVersion Conflicts`
`DisplayAllAuthcode`	`DownloadOnly`	`DownloadPolicy`
`EventLogs` (installation component)	`ForceCOMRegistration`	`ForceReboot`
`ForceRebootIfLocked`	`ForceValidSignature`	`GlobalConfigSource`

IgnoreConnectionWindows	InstallationStatus RefreshPeriod	InstallerARPModify
InstallerARPRemove	LogInstallCheck	LogInstallFail
LogInstallPass	LogLevel (installation component)	LogModules (installation component)
LogNotRequiredCheck	LogUninstallFail	LogUninstallPass
LowProfile (installation component)	MgsMsiArgs	MinimumDCSpeed
MsiAllPkgBaseURL	MsiBaseURL	MsiBaseURLList
MsiCustomBaseURL	MsiInstallArgs	MsiReinstallFeatures
MsiReinstallModeLevel	MsiRepair	MsiRepairLevel
MsiSourceListUpdate	MsiSourceLocation	MsiUILevel
MsiUninstallArgs	MsiUserDomain	NetworkRetries
NetworkRetryPeriodIncrement	NetworkTimeout (installation component)	NoStage
PkgCacheDirectory	PlatformSpecificPackages	PolicyPackageRefreshPeriod
PolicyRefreshPeriod	PolicyServerPriority	PostponeByDefault
PostponementQueryBefore	PostponeUserInteractionLevel	PrivateAppAccess
PromptOnCOMRegFailures	PromptOnInstallCompletion	PromptOnUnInstallCompletion
PropagatePkgChanged	PublicAppAccess	QuietUntilUpdate
RebootCmdLine	RebootIfRequired	RebootContinueAfterCmdFailure
RebootPostCommand	RebootPreCommand	RebootPromptCycles
RebootPromptWait	ReInstallRequires VersionChange	RenotifyTimeout
RotateConnectionsOnFailedConnectionAttempt	RunAllUsersInstallAsUser	SaveAllUserSymbols
SecurityAnalysisFile	ServiceConnectTimeout	StagedInstall
StagedOnly	StageInactivePackages	SupplyWorstCaseReturnValue
UITimeoutWait	UploadEventLogs	UseLastUpdateLocation
UseTrustDatabase	VerifyCatalogSigned	VerifyDownload
VerifyFilesSigned	VerifyTrustOrSign	VirusScan
VirusScanCommand		

ndschedag Command Line

Schedules are created on the central application server, and distributed to inventory beacons from which they are retrieved by managed devices. Managed devices then install the schedules, which are later read by the managed device's scheduling component.

Tip: *Listings of scheduled tasks are contextual to the privileges of the credentials running the schedule query. To see a full list of scheduled activities, be certain to choose the* ***Run as administrator*** *option when starting the Windows command window. Even if you are already logged in using an account that is an administrator on the computer, use this option, as Microsoft Windows by default runs some commands with lower privilege levels.*

Synopsis

Syntax:
ndschedag [*options*...]

Options:

Syntax:
-A
-e (non-Windows devices only)
-o *tag* = *value*
-x *eventId* (non-Windows devices only)

-A		Run all events of type `Schedule Update` in the installed schedule.
-e		Non-Windows devices only. Display a list of all scheduled events and their next run time.
-o	*tag*=*value*	Advanced parameter, recommended for use by experienced administrators only. Each instance over-rides the specified preference for the schedule component. Each parameter set at the command line must be accompanied by its own `-o` flag. Do not repeat any individual tag within the command line. Possible tags are listed below.
-x	*eventId*	Available on non-Windows devices only. Run the specified event. To identify the *eventId*, open the file `/var/opt/managesoft/scheduler/schedules/sched.nds` with a text editor. Find the line that defines the relevant event in the file, and copy the *eventId* associated with the event. Paste the *eventId* on the command line.

Example: Command line example
The following command runs the event with the ID {25ec6994-8f40-4985-bf4c-d57566c707c3}:

```
ndschedag -x "{25ec6994-8f40-4985-bf4c-d57566c707c3}"
```

Options

Possible tags for use with the `-o` options are:

- Catchup
- DisablePeriod (Windows devices only)
- MachineScheduleDirectory
- ndsensNetType (Windows devices only)
- OnConnect (Windows devices only)
- ScheduleType (Windows devices only)
- Startup
- UIMode (Windows devices only)
- UserScheduleDirectory (Windows devices only)

ndtrack Command Line

This is the command line reference for the tracker (the executable `ndtrack`). There is large overlap between use of the tracker in all its use cases, and specifically between the FlexNet inventory agent case and the FlexNet Inventory Scanner case. For this reason, variations are noted below, and you should use this reference for all cases.

Synopses

Syntax:
FlexNet inventory agent on Microsoft Windows:
`ndtrack.exe [options...]`

FlexNet Inventory Scanner on Microsoft Windows:
`FlexeraInventoryScanner.exe [options...]`

FlexNet inventory agent on UNIX-like platforms:
`ndtrack [options...]`

FlexNet Inventory Scanner on UNIX-like platforms:
`ndtrack.sh [options...]`

Options:

Syntax:
`-o` *tag* = "*value*"
`-t Machine`

Note: *The `-t Machine` option is mandatory for, and valid only for, a single use case: when you are running FlexNet*

inventory agent on Microsoft Windows from a user account other than the local `SYSTEM` account.

- *On UNIX-like platforms, it is always ignored.*
- *When running the lightweight FlexNet Inventory Scanner on Microsoft Windows, it is the default value and must not be specified in the command line.*
- *When executing `ndtrack.exe` on Microsoft Windows:*
  - *As the local `SYSTEM` account, it is the default.*
  - *As any other account, it must be specified.*

Any parameters declared on the command line override the default tracker settings. (On UNIX-like platforms in the FlexNet Inventory Scanner case [using `ndtrack.sh`], command-line parameters override both the default settings and any preferences recorded in `ndtrack.ini`.)

Tip: *The command line is processed left-to-right. This means that, where overlapping parameters are declared within the command line, the last one declared takes effect. For example:*

```
FlexeraInventoryScanner.exe -o Upload="false" -o Upload="true"
```

means that upload of collected inventory is attempted. In general, do not repeat any individual tag within the command line.

Possible tags are listed below (and see also the notes). Double quotation marks enclosing values are optional, except in the following cases where they are mandatory:

- Surrounding any value that includes white space
- Surrounding a list with internal semi-colon separators
- On Windows platforms using the `FlexeraInventoryScanner` executable, where `Upload="true"` is mandatory for normal operation, and the value *must* be enclosed in double quotation marks.

In general, special characters (double quotation marks, backslash) must be escaped with a backslash. However, `ndtrack` (*alone* of all the components documented in this chapter) adds special conditions:

- A backslash within a recognized file path need not be escaped.
- Avoid a backslash as the *last* character in a longer string enclosed in double quotation marks. For example, this option on an `ndtrack` command line *fails*:

  ```
  -o ExcludeDirectory="D:\;E:\;F:\;G:\"
  ```

  Here the final backslash-quote sequence is problematic. A general workaround is to insert a semicolon (`";"` which is the list separator character, usable only in the command line but not in PowerShell) between the backslash and double quotation mark. All of the following examples are successful:

  ```
  -o ExcludeDirectory="D:\;E:\;F:\;G:\;" // closing semi-colon avoids the problem
  -o ExcludeDirectory="D:\;" // works for a single item, as well as a list
  -o IncludeDirectory="\\" // short string also works, but beware huge inventory!
  -o IncludeDirectory=\\ // also works without quotes (but not for ExcludeDirectory)
  ```

Tip: *When troubleshooting exclusions, don't overlook the interaction between ExcludeDirectory and ExcludeEmbedFileContentDirectory.*

- You are unlikely to meet these problems on UNIX-like platforms, as the forward slash need not be escaped.

Return codes

The tracker returns a zero on success. If you receive a non-zero return code, check the log file. Details of the log file may also be configured with command-line options, as described in the section on Preferences:

- LogFile (inventory component)
- LogLevel (inventory component)
- LogModules (inventory component).

Example: Command line examples

This example collects a computer inventory and stores it locally (on the computer device where the FlexNet inventory agent is executing) for upload by a separate system (assuming execution by a non-`LocalSystem` account):

```
ndtrack.exe
          -t Machine
          -o MachineZeroTouchDirectory="local-folder"
          -o Upload=False
```

Tip: *An additional inventory (`.ndi`) file may be generated when all of the following conditions are true:*

- *You have licensed the FlexNet Manager for Datacenters product.*
- *The invoking account is correctly configured (for details, see Oracle Discovery and Inventory in FlexNet Manager Suite System Reference).*
- *Your current `InventorySettings.xml` file is correctly located for the tracker. Correct location depends on the particular case:*
  - *For the Adopted case and Agent third-party deployment case, in the folder identified in the `InventorySettingsPath` preference (defaults are `$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Tracker\InventorySettings\` on Windows and `/var/opt/managesoft/tracker/inventorysettings` on UNIX-like platforms). This is handled automatically for devices where the installed agent is managed by policy.*
  - *For the FlexNet Inventory Scanner case, in the same folder as the self-installing executable on Windows or the `ndtrack.sh` script for UNIX-like platforms,*
  - *For the Core deployment case, deployed into the same folder as the `ndtrack` executable.*
- *An Oracle Database is found on the target (local) device.*

The following example collects inventory and uploads it to an inventory beacon (assuming execution by the `LocalSystem` account). `ManageSoftRL` is the name of a web service on the inventory beacon that receives the uploaded inventory and saves it by default to *%CommonAppData%*`\Flexera Software\Incoming\`

Inventories:

```
ndtrack.exe -o UploadLocation="http://InventoryBeacon/ManageSoftRL"
```

Same purpose on a UNIX-like platform:

```
ndtrack -o UploadLocation="http://InventoryBeacon/ManageSoftRL"
```

Same purpose using FlexNet Inventory Scanner on Microsoft Windows (for this executable on this platform, the `-o Upload="true"` option, including the use of the double quotation marks, is mandatory):

```
FlexeraInventoryScanner.exe
    -o UploadLocation="http://InventoryBeacon/ManageSoftRL"
    -o Upload="true"
```

And same purpose using `ndtrack.sh` as a scanner on UNIX-like platforms:

```
ndtrack.sh -o UploadLocation="http://InventoryBeacon/ManageSoftRL"
```

The following command line is used for high-frequency (30 minute) hardware scanning when FlexNet Manager Suite is used for subcapacity license calculations for IBM PVU licenses:

```
ndtrack.exe -o WMI=true
            -o Hardware=true
            -o ManageSoftPackages=false
            -o MSI=false
            -o PlatformSpecificPackages=false
            -o Software=false
            -o TrackProductKey=false
            -o IncludeRegistryKey=
            -o IncludeDirectory=
            -o EmbedFileContentDirectory=
            -o OnlyGenerateIfHardwareChanged=true
            -o PerformSymantecSFScan=false
            -o PerformIBMWebSphereMQScan=false
            -o InventorySettingsPath="
```

Notes

1. Default values apply when a parameter is not specified.
2. If no drive is indicated when specifying directory paths, the tracker applies the path to every fixed drive of the local computer.
3. The tracker supports all name/value combinations as command-line options, although a warning is logged if a preference is used that does not appear in the list below.
4. A preference value can symbolically refer to another supported preference by enclosing its name thus: `$(`*`preferenceName`*`)`. References can contain further references.
   Example: The command

```
ndtrack.exe -o IncludeDirectory=$(WinDirectory)
```

includes the Windows directory in the scan (which is likely to produce a massive increase in file evidence!). References are resolved after all preferences are loaded so there are no ordering issues. Self-evidently, on UNIX-like platforms, only supported preferences can be referenced.

5. Semicolon or comma-separated values are the only method for defining multiple values in preferences for the tracker. (Do not repeat any individual tag within the command line.)
Example: The command

```
ndtrack.exe -o IncludeDirectory=C:\\;D:\\;E:\\
```

will scan the computer's C:, D:, and E: drives if they are fixed (hard) disks, but not if they are CD-ROM drives or logical drives (mapped to network locations).

6. To activate all logging, use the default preferences as follows:

```
Logging=true
LogLevel=A-Z
LogFile=<file>
LogModules=default
```

Options

Supported options are listed in the table below. A "Y" in columns 2-5 indicate support in:

- The full FlexNet inventory agent on Microsoft Windows, where preferences may also be included in the Windows registry (these same command-line settings may also be used if you have deployed just the FlexNet inventory core components in the Core deployment case, but in this case there is no checking of the Windows registry)
- The full FlexNet inventory agent on UNIX-like platforms (where preferences may also be included in the `config.ini` file)
- The lightweight FlexNet Inventory Scanner on Microsoft Windows (preferences can only be used in the command line)
- The scanner equivalent `ndtrack.sh` on UNIX-like platforms (where preferences may also be included in the `ndtrack.ini` file).

Possible tags for use with the `-o` options are:

Preference	Windows full agent	UNIX ndtrack	Windows Scanner	UNIX ndtrack.sh
AddClientCertificateAndKey		Y		
CALInventory	Y		Y	
CALInventoryPeriod	Y		Y	
CheckCertificateRevocation	Y	Y	Y	
CheckServerCertificate	Y	Y	Y	

Preference	Windows full agent	UNIX ndtrack	Windows Scanner	UNIX ndtrack.sh
ComputerDomain	Y	Y	Y	Y
DateTimeFormat	Y	Y	Y	Y
EmbedFileContentDirectory	Y	Y	Y	Y
EmbedFileContentExtension	Y	Y	Y	Y
EmbedFileContentMaxSize	Y	Y	Y	Y
ExcludeDirectory	Y	Y	Y	Y
ExcludeEmbedFileContentDirectory	Y	Y	Y	Y
ExcludeExtension	Y	Y	Y	Y
ExcludeFile	Y	Y	Y	Y
ExcludeLocalScriptRule	Y	Y	Y	Y
ExcludeMD5	Y	Y	Y	Y
GenerateMD5	Y	Y	Y	Y
Hardware	Y	Y	Y	Y
IncludeDirectory	Y	Y	Y	Y
IncludeExecutables	Y	Y	Y	Y
IncludeExtension	Y	Y	Y	Y
IncludeFile	Y	Y	Y	Y
IncludeMachineInventory	Y		Y	
IncludeLocalScriptRule	Y	Y	Y	Y
IncludeMD5	Y	Y	Y	Y
IncludeRegistryKey	Y		Y	
InventoryFile	Y	Y	Y	Y
InventoryScriptsDir	Y	Y	Y	Y
LogFile (inventory component)	Y	Y	Y	Y
LogLevel (inventory component)	Y	Y	Y	Y
LogModules (inventory component)	Y	Y	Y	Y
LowProfile (inventory component)	Y	Y	Y	Y
MachineInventoryDirectory	Y	Y		

Preference	Windows full agent	UNIX ndtrack	Windows Scanner	UNIX ndtrack.sh
MachineName	Y	Y	Y	Y
MachineZeroTouchDirectory			Y	
MSI	Y	Y	Y	Y
NetworkHighSpeed	Y	Y	Y	Y
NetworkHighUsage	Y	Y	Y	Y
NetworkHighUsageLowerLimit	Y	Y	Y	Y
NetworkHighUsageUpperLimit	Y	Y	Y	Y
NetworkLowUsage	Y	Y	Y	Y
NetworkLowUsageLowerLimit	Y	Y	Y	Y
NetworkLowUsageUpperLimit	Y	Y	Y	Y
NetworkMaxRate	Y	Y	Y	Y
NetworkMinSpeed	Y	Y	Y	Y
NetworkSense	Y	Y	Y	Y
NetworkSpeed	Y	Y	Y	Y
OracleEnvironmentCmdTimeoutSeconds		Y		Y
OracleInventoryAsSysdba		Y		Y
OracleInventoryUser		Y		Y
PerformKvmInventory		Y		Y
PerformLocalScripting	Y	Y	Y	Y
PerformOracleFMWScan	Y	Y	Y	Y
PerformOracleInventory	Y	Y	Y	Y
PerformOracleListenerScan	Y	Y	Y	Y
PreferIPVersion	Y	Y	Y	Y
PrioritizeRevocationChecks		Y	Y	
ProgramFiles, ProgramFilesX86Folder, ProgramFilesX64Folder	Y	Y		
Recurse	Y	Y	Y	Y
RunInventoryScripts	Y	Y		
ShowIcon (inventory component)	Y	Y		

Preference	Windows full agent	UNIX ndtrack	Windows Scanner	UNIX ndtrack.sh
SSLCACertificateFile		Y		
SSLCACertificatePath		Y		
SSLClientCertificateFile		Y		
SSLClientPrivateKeyFile		Y		
SSLCRLCacheLifetime		Y		
SSLCRLPath		Y		
SSLDirectory		Y		
SSLOCSPCacheLifetime		Y		
SSLOCSPPath		Y		
SysDirectory	Y	Y		
UploadLocation	Y	Y	Y	Y
VersionInfo	Y	Y		
WinDirectory	Y		Y	
WMI	Y		Y	
WMIConfigFile.	Y		Y	
Upload (note separate defaults)	Y	Y	Y	Y

ndupload Command Line

The uploader runs on managed devices and inventory beacons. It allows you to transfer event logs, inventories and other files from managed devices to FTP or local file servers.

Tip: *Only HTTP and HTTPS protocols are supported by inventory beacons. File and FTP protocols are available for alternative upload arrangements.*

The uploader can transfer any file to a specified URL. If an FTP URL is supplied, then a username and password must also be given.

The uploader deletes files from managed devices after they have been successfully uploaded. Files are not removed if the upload fails.

The file path supplied to the uploader can contain wildcards, so that multiple files of a similar type can be uploaded with a single command.

Synopsis

Syntax:
ndupload.exe [*options*...]

Options:

Syntax:
-a
-f *path\and\filename.ext*
-o *tag* = *value*

-a		This is the same as `-o UploadRule` and means "upload all file types". If you run `ndupload` with no command line parameters, this is the default behavior.
-f	*path\and\filename.ext*	Identifies the file to upload. This is the same as `-o SourceFile`.
-o	*tag*=*value*	Each instance over-rides the specified preference for the uploader. Each parameter set at the command line must be accompanied by its own `-o` flag. Do not repeat any individual tag within the command line. Possible tags are listed below.

Return codes

The uploader returns a zero on success. If you receive a non-zero return code, check the log file. Details of the log file may also be configured with command-line options, as described in the section on Preferences:

- LogFile (upload component)
- LogFileOld (upload component)
- LogFileSize (upload component).

Example: Command line examples

The following command uploads the file `file.txt` to the FTP location `ftp://server/dir1/dir2` using the login name `user1` and the password `abc123`:

```
ndupload -f file.txt -o UploadLocation=ftp://server/dir1/dir2
-o UploadUser=user1 -o UploadPassword=abc123
```

The following command uploads the file `file.txt` to the mapped drive `f:/dir1/dir2`:

```
ndupload -f file.txt -o UploadLocation=file:///f:/dir1/dir2
```

This example uploads the inventory file `myInventory.ndi` to an inventory beacon (where `ManageSoftRL` is the name of a web service on the inventory beacon that receives the uploaded inventory and saves it by default to *%CommonAppData%*`\Flexera Software\Incoming\Inventories`):

```
ndupload -f myInventory.ndi -o
UploadLocation="http://InventoryBeacon/ManageSoftRL"
```

Options

Tip: *Although the* `Network*` *preferences remain available for special circumstances, network throttling for package downloads is not normally required. For details, see the discussion under NetworkSpeed.*

Possible tags for use with the `-o` options are:

- AddClientCertificateAndKey (UNIX-like platforms only)
- CheckCertificateRevocation
- CheckServerCertificate
- LogFile (upload component)
- LogFileOld (upload component)
- LogFileSize (upload component)
- MaxKeepAliveLifetime
- MaxKeepAliveRequests
- NetworkHighSpeed
- NetworkHighUsage
- NetworkHighUsageLowerLimit
- NetworkHighUsageUpperLimit
- NetworkLowUsage
- NetworkLowUsageLowerLimit
- NetworkLowUsageUpperLimit
- NetworkMaxRate
- NetworkMinSpeed
- NetworkSense
- NetworkSpeed
- NetworkTimeout
- PreferIPVersion
- PrioritizeRevocationChecks (UNIX-like platforms only)
- SendTCPKeepAlive (Windows platforms only)
- SourceFile
- SourceRemove
- SSLCACertificateFile (UNIX-like platforms only)

- SSLCACertificatePath (UNIX-like platforms only)
- SSLClientCertificateFile (UNIX-like platforms only)
- SSLClientPrivateKeyFile (UNIX-like platforms only)
- SSLCRLCacheLifetime (UNIX-like platforms only)
- SSLCRLPath (UNIX-like platforms only)
- SSLDirectory (UNIX-like platforms only)
- SSLOCSPCacheLifetime (UNIX-like platforms only)
- SSLOCSPPath (UNIX-like platforms only)
- `TenantUID`, to override the value set in the registry (multi-tenant mode only)
- UploadLocation
- UploadPassword
- UploadRule
- UploadType
- UploadUser.

10

Preferences

This section contains an alphabetic listing of some useful preferences you can declare on agent command lines (for example, during testing or in scheduled tasks) or store in the registry for run-time assessment.

Note: *Preference defaults are provided, where appropriate, for Windows and Unix platforms.*

[Registry] Explained

The following definitions of computer preferences use the placeholder `[Registry]`. This text represents the location of all relevant registry entries in the registry:

- On Windows servers, inventory beacons, and managed devices, agent registry entries on 32-bit operating systems are usually stored under the key:

  ```
  HKEY_LOCAL_MACHINE\SOFTWARE\ManageSoft Corp\
  ```

  For 64-bit operating systems, the normal key is:

  ```
  HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ManageSoft Corp\
  ```

  For these platforms, the `[Registry]` placeholder should be interpreted as the appropriate one of these keys.

  Tip: *Equivalent keys in the HKEY_CURRENT_USER hive are now deprecated (meaning that their use is discouraged, that their ongoing functionality is not guaranteed, and that at an unspecified future time support for this hive may be entirely removed).*

- On non-Windows managed devices, registry entries are stored in the `/var/opt/managesoft/etc/config.ini` file. Within this file, registry keys are shown in square brackets. The lines below each key show the registry entries set under that key. For example, to set the type of inventory to be collected as a 'registry' preference on UNIX-like managed devices:

  ```
  [ManageSoft\Tracker\CurrentVersion]
  InventoryType=Machine
  ```

  In other words, on UNIX-like devices, the documentation placeholder `[Registry]\` should be taken to mean "look

in the `/var/opt/managesoft/etc/config.ini` file for the following key".

Absent keys

Not all registry keys cited in the following preferences exist by default. These are usually marked "Code internals, or manual configuration". If the registry key does not exist, or if it exists but has no value, the default value is provided by the FlexNet inventory agent (without modifying the registry). However, if you wish to override the default value for the preference, you should manually create the registry key shown, and populate it with your alternate value.

AddClientCertificateAndKey

Command line | Registry

When using the HTTPS protocol for any communication between a managed inventory device (the client) and an inventory beacon (the server), the communication is secured by one of two kinds of Transport Layer Security (TLS):

- In unilateral or standard TLS, the server has a valid certificate and a public/private key pair (but the client does not). To be valid, a certificate must have been issued by a Certificate Authority that is also trusted by the client (and the DNS name on the certificate of course matches the DNS name of the server). When the client connects to the server, the server presents its TLS certificate, and the client verifies the server's certificate. If the certificate is verified successfully, the communication from this point is done on an encrypted TLS connection.
- In mutual TLS, both the client and server have valid certificates, and both sides authenticate using their public/private key pairs:
  1. When the client connects to the server, the server presents its TLS certificate and the client verifies the server's certificate.
  2. Now the client presents its TLS certificate, and the server verifies the client's certificate.
  3. If both certificates are verified successfully, the communication is done on an encrypted TLS connection.

Keep in mind that the FlexNet inventory agent has multiple components that may either receive or send communications from/to an inventory beacon, such as the installation component, the inventory component, and the upload component. This means that for mutual TLS, all components of the FlexNet inventory agent must be able to provide the client certificate. All the client components make use of this `AddClientCertificateAndKey` preference, which is disabled by default, and must be enabled to allow use of mutual TLS. There is a `Common` preference available, so that the setting applies to all components; and, if necessary you can override the common behavior with settings for individual components. You can also set the individual preferences to the same value, which may provide more reliable operation.

Tip: *As well as setting the `AddClientCertificateAndKey` preference for all required clients (managed devices where the FlexNet inventory agent is locally installed, and communicating routinely with one or more inventory beacons), the inventory beacon server must also be configured to require a client-side certificate for authentication in mutual TLS. Be aware that this is a single setting on the inventory beacon, so that once an inventory beacon is configured for mutual TLS with a single client, it requires mutual TLS from every FlexNet inventory agent. Since each installation of the FlexNet inventory agent may randomly choose which inventory beacon to contact (for example, for policy updates, or for uploads of collected inventory), this means that the decision to use mutual TLS is a global one to be implemented across (at least) an entire partition of your network.*

Values

Values / range	Boolean (`True` or `False`)
Default value	`False`
Example values	`True`

Command line

Tool	Installation component (`ndlaunch`), inventory component (`ndtrack`), upload component (`ndupload`)
Example	`-o AddClientCertificateAndKey="True"`

Registry

Installed by	Manual configuration

Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`
	`[Registry]\ManageSoft\Tracker\CurrentVersion`
	`[Registry]\ManageSoft\Uploader\CurrentVersion`
	`[Registry]\ManageSoft\Common`

AllowedPkgTypes

Command line | Registry

`AllowedPkgTypes` is a semi-colon (`;`) separated list of installable package types. Whereas the policy for the FlexNet inventory agent references all the packages (of all kinds) that are available for this managed device, the `AllowedPkgTypes` setting acts as a filter, so that only the listed types are included in policy updates. An empty value allows installation of all package types. Clearly this preference is intended for internal use, but may on rare occasions be useful for testing.

The package types available are:

- `Package` (normal FlexNet packages)
- `ClientSettings` (only packages containing fail-over settings)
- `ClientConfiguration` (only packages containing managed device settings)
- `Schedule` (only packages containing details of scheduled events).

Values

Values / range	`Package`, `ClientSettings`, `ClientConfiguration`, `Schedule`
Default value	(No default.)
Example values	`ClientSettings;Schedule`

Command line

Tool	Installation component (`ndlaunch`), policy component (`mgspolicy`)
Example	`-o AllowedPkgTypes="ClientConfiguration"`

Registry

Installed by	Code internals
Computer preference	`[Registry]\ManageSoft\Common`

Application

Registry

`Application` identifies the name of an application in the manual mapper for the application usage component. After usage is detected and uploaded, this name appears in the **Raw Software Usage** page in the web interface for FlexNet Manager Suite. This registry key must be created manually, within a node for the chosen application that has also been inserted manually (and shown below as *`Application node`*).

Tip: *If this value for `Application`, together with the value for `Version`, are exact matches for the application name and version recorded in installer evidence for the application, the set-up can be simplified a little, because this mapping of the executable to the application automatically matches through the known installer evidence. (If not, this manual mapper entry can also be manually linked to the application record.)*

Values

Values / range	Valid text character string, which may include spaces as required.
Default value	No default value.
Example values	`Microsoft Project`

Registry

Installed by	Manual configuration only.
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion\Manual Mapper\`*`Application node`*

AutoPriority

Registry

`AutoPriority` determines whether an inventory beacon has a priority calculated at the time of download or upload (`True`), or whether it is fixed at the value declared in the `Priority` registry key (`False`).

Priorities are used to determine the order in which connections to reporting locations or distribution locations are attempted. If `AutoPriority` does not exist under a download or upload location's registry settings, the appropriate agent assumes the value `True`. If you want to use fixed priorities, you must create the `AutoPriority` registry key if it does not exist, and assign to it the value `False`.

Values

Values / range	Boolean (`True` or `False`)
Default value	`True`

Registry

Installed by	Failover list for inventory beacons, or manual configuration

Computer preference	For uploads:
	`[Registry]\ManageSoft\Common\UploadSettings\<reporting_location>`
	For downloads:
	`[Registry]\ManageSoft\Common\DownloadSettings\<distribution_location>`

CALInventory

Command line | Registry

`CALInventory` determines whether or not the FlexNet inventory agent attempts to collect access evidence from the

managed device in a `.swacc` file.

Tip: *The `.swacc` file is saved on the managed device in `...\Uploads\ClientAcess` (the default path is `C:\ProgramData\ManageSoft Corp\ManageSoft\Common\Uploads\ClientAccess`). This location cannot be modified. Because the data here is strongly time-related, the file is removed after it has been uploaded to an inventory beacon.*

Values

Values / range	Boolean (`True` or `False`)
Default value	`False`
Example values	`True`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o CALInventory=True`

Registry

Installed by	A downloaded client settings package, or manual setting (computer preference)
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

CALInventoryPeriod

Command line | Registry

`CALInventoryPeriod` sets the interval (in days) between times when FlexNet inventory agent saves a `.swacc` file on the managed device.

Tip: *If `CALInventory` is false (the default), `CALInventoryPeriod` is ignored.*

Values

Values / range	Zero or a positive integer

Default value	`90` (This is the default value when there is no entry in the registry or command line options.)
Example values	`7`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o CALInventoryPeriod=10`

Registry

Installed by	Manual configuration (computer preference)
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

Catchup

Command line | Registry

`Catchup` controls the behavior of events occurring on the managed device. It determines when the FlexNet inventory agent will catch up missed scheduled events. Possible actions include:

- `Always`: the agent always attempts to catch up on all missed events.

  Note: *With this value set, running the scheduling component does not display its graphical interface. This value is used temporarily during self-upgrades of the FlexNet inventory agent to ensure that no events that were scheduled to occur during the upgrade are missed, and to confirm the success of the upgrade. For this reason, it cannot display a user interface, since that would block the self-upgrade while waiting for a user response.*

- `Never`: the agent never attempts to catch up on missed events.
- `Once`: the agent attempts to catch up on only the most recently missed event. For example, if the inventory device was powered down at the time when there was an event scheduled for the machine, this setting causes the missed event to run shortly after the inventory device is powered up again.

  Tip: *This setting is used if you select the* ***Run last missed event*** *control in the* ***Inventory agent schedule*** *section of the* ***Inventory Settings*** *page.*

- `Query`: the agent queries the user regarding catch up on missed events (available for Windows devices only).

Values

Values / range	`Always`, `Never`, `Once`, `Query`
Default value	No default in registry; default behavior is `Never`
Example values	`Always`

Command line

Tool	Scheduling component (`ndschedag`)
Example	`-o Catchup=Query`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)
Computer preference	`[Registry]\ManageSoft\Schedule Agent\CurrentVersion`

CheckCertificateRevocation

Command line | Registry

When transferring data to or from an inventory beacon using the HTTPS protocol, a web server certificate is applied to the data being transferred.

When receiving web server certificates from servers, the appropriate client-side component checks the CA (certification authority) server to ensure that the certificates are not on the CRL (certificate revocation list). If a component cannot check the CRL (for example, the CA server is firewalled and cannot be contacted), the system may time out on the CRL download, and consequently fail the revocation check. To avoid this, you can use the `CheckCertificateRevocation` preference to prevent components from performing the CRL check.

Tip: *Turning off CRL checking should be only a temporary measure while you fix the problem that prevents successful checking. It is poor security practice to omit the check for certificate currency, since without this check your system may continue to trust a certificate that has been compromised as part of a wider attack.*

You can set this as a common registry entry, so that the same behavior occurs across all components, and you can override the common behavior by setting an overriding registry entry for any individual component if required. By default, this preference is set so that all components check the CRL.

Values

Values / range	Boolean (`True` or `False`)
Default value	`True`
Example values	`False`

Command line

Tool	`ndtrack`, `ndlaunch`, `ndupload`
Example	`-o CheckCertificateRevocation="False"`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Common` or `[Registry]\ManageSoft\<Component>\CurrentVersion` where *<Component>* is the registry key for an individual component (`Launcher`, `Tracker`, or `Uploader`). ***Note:*** *In some circumstances only the specific path for a component works.*

CheckServerCertificate

Command line | Registry

When transferring data to or from an inventory beacon using the HTTPS protocol, a web server certificate is applied to the data being transferred. All components can (and by default do) validate the public certificates received from the inventory beacon against their local copy (on Windows, in the certificate store; and on UNIX in the PEM file).

If you wish, you can use the `CheckServerCertificate` preference to prevent agents from performing the certificate check. (Without this check, the certificate is ignored, and the HTTPS protocol provides only encryption as security on the transfer, without validating that the agent is contacting the correct inventory beacon server.)

You can set this as a common registry entry, so that the same behavior occurs across all components; and you can override the common behavior by setting an overriding registry entry for any individual component if required. By default, this preference is set so that all components check the inventory beacon server certificate against the root CA certificate.

Values

Values / range	Boolean (`True` or `False`)
Default value	`True`
Example values	`False`

Command line

Tool	`ndtrack`, `ndlaunch`, `ndupload`
Example	`-o CheckServerCertificate="False"`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Common` or `[Registry]\ManageSoft\<Component>\CurrentVersion` where *<Component>* is the registry key for an individual component. ***Note:*** *In some circumstances, only the more specific path for a specific component works.*

CommonAppDataFolder

Command line | Registry

`CommonAppDataFolder` provides the path to the folder in which application details are located. On Windows managed devices, these are application details for `[ALL USERS]`. This is a system variable which you can override in the registry or on the command line.

Values

Values / range	Local directory path. Read-only preference.

Default value	The default installation of Windows uses:
	`%ALLUSERSPROFILE%\Application Data`
	where %ALLUSERSPROFILE% defaults to:
	• On Windows 2000/XP: `C:\Documents and Settings\All Users`
	• On Windows Vista: `C:\ProgramData`
	• On Windows 7 and higher: `C:\Users\Public`
	For Macintosh and UNIX devices, the value is:
	`/var/opt/managesoft`

Example values	Windows:
	`C:\Users\Public\Application Data`
	UNIX-like systems:
	`/var/opt/bin`

Variable

Defined:	Predefined within Windows.
Reference as:	`$(CommonAppDataFolder)`

Compress (application usage component)

Command line | Registry

`Compress` specifies whether or not application usage data files are compressed before being uploaded through your inventory beacon(s) to the administration server for inclusion, initially, in the inventory database:

- When set to `True`, the usage component uses `gzip` to compress the application usage data file for upload, and the file name format is *`deviceName`* `at` *`timestamp`*`.mmi.gz`
- When set to `False`, the file is left uncompressed, and the file name format is *`deviceName`* `at` *`timestamp`*`.mmi`.

In the file names:

- *`deviceName`* is the computer name of the inventory device
- *`timestamp`* is the date and time when the `.mmi` file was saved.

The files are saved:

- On Windows devices, in `C:\ProgramData\ManageSoft Corp\ManageSoft\Common\Uploads\UsageData`

- On UNIX-like devices, in `/var/opt/managesoft/uploads/UsageData`.

The files are removed after a successful upload.

Values

Values / range	Boolean (true or false)
Default value	`True`
Example values	`False`

Command line

Tool	Application usage agent (`mgsusageag`)
Example	`-o Compress=False`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

ComputerDomain

Command line | Registry

`ComputerDomain` contains the canonical name of the domain of the local computing device, from which inventory is being gathered. One reason for setting this value is to cause UNIX-like devices to report in a particular domain in listings and reports within FlexNet Manager Suite.

Note: *You can configure this setting for deployment to UNIX-like devices by setting the `MGSFT_DOMAIN_NAME` property in the `mgsft_rollout_response` file. Alternatively, you can manually edit the value in the `/var/opt/managesoft/etc/config.ini` file that serves as a registry store on UNIX-like devices where the FlexNet inventory agent is installed. For more information, see Agent Third-Party Deployment: Configuring the Bootstrap File for UNIX.*

Values

Values / range	Valid domain name.

Default value	On Windows, the current local Active Directory domain for Windows devices. For UNIX-like devices, the default value is *$(OSComputerDomain)*.
Example value	`MyDomain.com`

Command line

Tool	Inventory agent
Example	`-o ComputerDomain=MyDomain.com`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Common`

ConnectionAttempts

Command line | Registry

`ConnectionAttempts` applies only to the use of the `file:` protocol for uploads/downloads in a custom implementation. For standard protocols used by inventory beacons, see NetworkRetries.

When the installation tool within the FlexNet inventory agent is trying to connect with a file share using the `file:` protocol, `ConnectionAttempts` specifies how many times it will accept a "no connection is available" error before discarding that location and logging a network failure. The "no connection is available" condition occurs when the number of active connections to a file share reaches the maximum allowed.

Values

Values / range	Numeric
Default value	`2`
Example values	`100`

Command line

Tool	Installation component (`ndlaunch`)

Example	`-o ConnectionAttempts=100`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`

DateTimeFormat

Command line | Registry

`DateTimeFormat` sets the date/time format for all inventory agent activity.

Warning: *Internal use only: do not edit.*

Values

Values / range	Date/time format definition string, based on the ANSI C function `strftime()`.
Default value	`%Y%m%dT%H%M%S`
Example value	`%I.%M.%S %p - %A, %d %B %Y` This would result in a date format such as `10.30.05 am - Monday, 26 February 2010` ***Note:*** *Since the DateTimeFormat string is also used as a file name, you may use only characters that are valid in file names across all platforms (in particular, you may not use a colon).*

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o DateTimeFormat="%H.%M.%S %p - %A, %d %B %Y"`

Registry

Installed by	Agent code, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

DefaultSchedulePath

Command line | Registry

`DefaultSchedulePath` gives the URL or path to the default machine schedule that is installed by the policy agent before applying policy if `InstallDefaultSchedule` is `True`.

Usually this preference is used to point to a schedule, but it can be configured to point to any `.osd` file that the policy agent wants the installation agent to process before applying policy.

Values

Values / range	Any valid directory path and filename
Default value	`$(DownloadRootURL)/Schedules/Default Machine Schedule/Default Machine Schedule.osd`
Example values	`C:\temp\debugSchedule.osd`

Command line

Tool	Policy component (`mgspolicy`)
Example	`-o DefaultSchedulePath=”C:\Schedules\Default Machine Schedule\Default Machine Schedule.osd”`

Registry

Installed by	Installation of FlexNet inventory agent
Computer preference	`[Registry]\ManageSoft\Policy Client\CurrentVersion`

Directory

Registry

`Directory` represents one of the following:

- For uploads, the folder representing the reporting location's upload folder
- For downloads, the location of the distribution location, within the specified host
- For trusted and excluded locations, the location of the distribution location, within the specified host.

Values

Values / range	Folder location
Default value	(No default.)
Example values	`/ManageSoftDL`

Registry

Installed by	Failover list for inventory beacons, or manual configuration

Computer preference	For uploads:
	`[Registry]\ManageSoft\Common\UploadSettings\<reporting_location>`
	For downloads:
	`[Registry]\ManageSoft\Common\DownloadSettings\<distribution_location>`
	For trusted locations:
	`[Registry]\ManageSoft\Launcher\CurrentVersion\TrustedLocations\<serverkey>`
	For excluded locations:
	`[Registry]\ManageSoft\Launcher\CurrentVersion\ExcludedLocations\<serverkey>`

DisableAllAgentUploads

Command line | Registry

When `DisableAllAgentUploads` is set to `True`, inventory files generated by components of the FlexNet inventory agent (such as the tracker or the usage agent) are saved locally on the local inventory device, but are *not* uploaded to any inventory beacon. This includes the (default nightly) catch-up transfers by the `ndupload` component.

Tip: *When* `DisableAllAgentUploads` *is set to* `True`*, it overrides the setting for the Upload preference (which, in any case, controls only the tracker).*

When `DisableAllAgentUploads` is `False`, normal upload operations apply (including allowing control by other preferences like `Upload` and DisablePeriod).

Restriction: *While* `DisableAllAgentUploads` *controls uploads from all components of the FlexNet inventory agent, it has no effect on uploads initiated by either the Flexera Kubernetes inventory agent or the lightweight Kubernetes agent. Furthermore, if the* `imgtrack` *container inventory tool is in use (and running* `ndtrack.sh` *on UNIX-like platforms), it does not have this preference set, and therefore continues the normal upload operations from that device.*

This setting can be used to temporarily turn off uploads from all components of the locally-installed FlexNet inventory agent to allow, for example, investigations of potential network security concerns.

Values

Values / range	Boolean (`True` or `False`)
Default value	`False` This enables normal upload operations from the FlexNet inventory agent to the inventory beacon of its choice.

Command line

Tool	Inventory component (`ndtrack`) and upload component (`ndupload`)
Example	`-o DisableAllAgentUploads=True` This allows you, for example, to run a test inventory collection on the local inventory device, but to not upload the resulting `.ndi` file, instead holding the file locally for examination.

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Common`

Disabled (application usage component)

Command line | Registry

`Disabled` specifies whether the application usage component is inactive on this managed device. When set to `True`,

the FlexNet inventory agent does not record application usage data. When set to `False`, the FlexNet inventory agent records application usage data.

Note: *The schedule component has a preference of the same name, used for a slightly different purpose.*

Background: Application usage tracking (sometimes called "application metering") by the FlexNet inventory agent works by tracking the time during which installed files are opened and active on the targeted devices, where those installed files are known to be part of a particular installed application. This usage data is uploaded to the central application server, where the usage tracking calculations occur at each license reconciliation (whether or not usage data is available from any particular inventory source). This preference can be modified in three ways:

- At adoption or installation time for the FlexNet inventory agent on a target Windows device, the installation configuration file can establish your preferred default (but only for Windows):
  - On Windows, the `USAGEAGENT_DISABLE` setting from the `mgssetup.ini` file is written to the registry location shown below
  - On UNIX-like platforms, the `mgsft_rollout_response` file does not support a setting for application usage tracking; but manual editing of the preference is possible (described next).
- On a target device, you can edit this preference setting manually:
  - On Windows, edit the registry setting shown below, or deploy a registry change using your preferred deployment tool
  - For UNIX-like platforms, you can edit the UNIX preference file (`config.ini`) and deploy the update separately.
- The simplest method is that you can update the setting for selected target devices through the web interface. Navigate to **Discovery & Inventory > Discovery and Inventory Rules > Targets** tab, and scroll down to the **Application usage options** section. Changes recorded here are deployed automatically to the target inventory devices through the next policy update, where they adjust this setting appropriately on each device.

Values

Values / range	Boolean (`True` or `False`)
Default value	`True`

Command line

Tool	Application usage agent
Example	`-o Disabled=False`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration

Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

Disabled (schedule component)

Command line | Registry

`Disabled` determines whether the schedule component is disabled on the managed device. By default, this value is set to `False`, enabling the schedule component.

If a date and time in the future is specified, the schedule component is disabled, and remains disabled until this date and time. Settings are not cleared as a given date and time passes, so that a date and time that has passed is also a possible value. Such past values are ignored, and have the same effect as the value `False` (that is, when the date is in the past, the schedule agent is enabled).

On Windows devices, management of this preference is different for user schedules and machine schedules:

- For user schedules, the user can run the schedule agent on the managed device. If users open a command line window, they can navigate to `$(ProgramFiles)\ManageSoft\Schedule Agent`, and run `ndschedag.exe`, without parameters. This presents a user interface where they can set the **Disabled** check box. When the schedule agent window closes (with the check box set), the schedule agent adds the number of seconds in the `DisablePeriod` preference to the current date and time, and writes the result to this `Disabled` (schedule agent) setting.

  Note: *User-based inventory (and therefore user-based scheduling) are deprecated, and are included only for backward compatibility. Use of machine schedules is recommended.*

- Machine schedules cannot be disabled through any user interface, nor through command line interaction. If you need to temporarily disable a machine schedule, enter an appropriate date and time string in ISO format in the *Computer preference* registry setting listed in the **Registry** table below.

Values

Values / range	Either `False`, or a date/time value in the ISO standard format *`yyyymmdd`*`T`*`hhmmss`*. If this date and time is in the future, it is the date and time at which the agent schedules will be re-enabled on this managed device.
Default value	`False`
Example values	`20160312T101520`

Command line

Tool	Schedule component (`ndschedag`)

Example	`-o Disabled=20150823T143014`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)
Computer preference	`[Registry]\ManageSoft\Schedule Agent\CurrentVersion`

DisablePeriod

Command line | Registry

`DisablePeriod` is only applicable when `Disabled` (schedule agent) is set to `True`. It sets the number of seconds for which agent schedules are disabled when the user selects "Disabled" from the scheduling agent user interface on the managed device. (The scheduling agent user interface is only available on Windows devices.)

The default value is 3600 seconds (one hour). When set to 3600, agent schedules are automatically re-enabled after one hour.

Values

Values / range	0 - 2147483647
Default value	`3600`
Example values	`1800`

Command line

Tool	Scheduling component (`ndschedag`)
Example	`-o DisablePeriod=600`

Registry

Installed by	Installation of FlexNet inventory agent
Computer preference	`[Registry]\ManageSoft\Schedule Agent\CurrentVersion`

DownloadSettings

Registry

`DownloadSettings` is a registry key (container) for several preferences that can control the download of data by the FlexNet inventory agent. These registry values are in sets that apply to a particular download location, for which reason the registry key must be completed with an identifier for the download location. The completed path leads to the relevant set of registry values, as shown below.

When configured by the failover list generated by an inventory beacon, the placeholder *<download_location>* in the registry path takes the form of a GUID that identifies the download location on the particular inventory beacon (for example, `{EEFC121D-2BB3-4318-8014-8DDC339C7553}`).

For manual configuration, four name/value pairs must be specified, and others are optional. To omit an optional value, you may include the name and leave the value blank (as shown in the example below), or omit the name/value pair entirely. The values that may be set are:

- `Protocol` – Mandatory. For download from an inventory beacon, this must be either `http` or `https`.
- `Name` – a user-friendly (general purpose) name for this group of settings. When set by policy downloaded from an inventory beacon, this consists of the value of Host with the string " `Download Location`" appended.
- `Directory` – Mandatory. By default, the download location is called `ManageSoftDL`, but as this may have been renamed during installation, you need to check details of your implementation.
- `Host` – Mandatory. When set by downloaded policy, this is normally the fully-qualified domain name of the inventory beacon. If you are setting this registry value manually, you may instead use either an IPv4 or an IPv6 (unique global or unique local) address.
- `Port` – Mandatory. As this has no default value, you must specify this setting to suit your environment (typically port 80 for HTTP and port 443 for HTTPS).
- `User` – If omitted (or left with a blank value), anonymous authentication is used for downloads from this download location.
- `Password` – Stores an encrypted copy of the password needed when Windows authentication is specified for the download.
- `Priority`
- `AutoPriority`
- Proxy.

Values

Values / range	Requires a subkey that uniquely identifies the download location on an individual inventory beacon.
Default value	None.

Example values	`[Registry]\ManageSoft\Common\DownloadSettings\{EEFC121D-2BB3-4318-8014-8DDC339C7553}`
	`Protocol=http`
	`Name=ss-server1 Download Location`
	`Directory=ManageSoftDL`
	`Host=ss-server1`
	`Port=80`
	`User=`
	`Password=`
	`Priority=10`
	`AutoPriority=True`

Registry

Installed by	Download of failover settings, or manual configuration
Computer preference	`[Registry]\ManageSoft\Common\DownloadSettings\<download_location>`

EmbedFileContentDirectory

Command line | Registry

`EmbedFileContentDirectory` specifies folders that must be scanned in the search for ISO/IEC 19770-2 software identification tags (known as "SWID tags", for which see https://tagvault.org/swid-tags/what-are-swid-tags/). Note that subfolders may be included, based on the value of `Recurse` (which defaults to true). When recursion is needed, specific subfolders may also be excluded (see ExcludeEmbedFileContentDirectory).

This preference may be set by downloaded policy to reflect the settings in the web interface for FlexNet Manager Suite at **Discovery & Inventory > Settings**.

Tip: *This is not the only preference that causes folders to be scanned. See also IncludeDirectory.*

Values

Values / range	Any valid folder path. Multiple paths may be specified with a semi-colon separator between them.

Default value	`"%ProgramData%;%ProgramFiles%;%PROGRAMFILES(x86)%"` This default applies on Windows platforms when the registry key is not present, and the option is not specified in the command line. ***Important:*** *If the registry key is present but has a null value (or the option is given in the command line but without a value), scanning for SWID tags is turned off.* ***Note:*** *On UNIX-like platforms, there is no default path scanned. To embed ISO tag files on these platforms, you may specify paths in the web interface(see above) or use this preference either on a command line or in:* • *For the FlexNet inventory agent case, the `/var/opt/managesoft/etc/config.ini` file that serves in place of a registry on these platforms* • *For the FlexNet Inventory Scanner case, the co-located `ndtrack.ini` file that holds preferences for `ndtrack.sh`.*
Example value	`"\\"` The double backslash works around the command prompt escaping, and becomes a single backslash to the inventory component. This searches all folders on all available drives for ISO tag files. ***Note:*** *This would be a time-consuming operation, and not best practice given that the ISO specification defines where ISO tag file should be stored (see the default value above).*

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o EmbedFileContentDirectory="C:\Program Files (x86)"`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

EmbedFileContentExtension

Command line | Registry

`EmbedFileContentExtension` defines the file extension(s) that must be matched for a small text file (by design, an ISO tag file) to be embedded within the uploaded inventory (`.ndi`) file.

Values

Values / range	Any valid file extension, excluding the leading dot or period character. Multiple file extensions may be included, separated by the semicolon character.
Default value	`swidtag` Default applies when no value is visible in the registry or command line. ***Note:*** *This default value excludes the ISO tag files for Adobe products (such as Adobe Acrobat 9 and Creative Suite 4) released around the time that the ISO 19770-2 standard was announced. These early implementations used a file extension of `swtag`. The following example changes the default to include both standard and early-adopter versions of the ISO tag file extension.*
Example values	`"swidtag;swtag"`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o EmbedFileContentExtension="swidtag;swtag"`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

EmbedFileContentMaxSize

Command line | Registry

`EmbedFileContentMaxSize` specifies an upper file size limit in bytes for text files (such as ISO tag files) that are to be

included/embedded in the uploaded inventory (`.ndi`) file.

Values

Values / range	Any valid integer, which is interpreted as bytes.
Default value	`1000000` This default is approximately equivalent to one megabyte.
Example values	`"1024"` This value limits the size of embedded files to one kilobyte.

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o EmbedFileContentMaxSize="1024"`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

ExcludeDirectory

Command line | Registry

Important: *This preference may be used manually in the command line on any platform. Any value set in the `[Registry]` may be overwritten each time that device policy is downloaded from an inventory beacon. Preferences for the installed and policy-managed FlexNet inventory agent follow your settings in the web interface for FlexNet Manager Suite at* ***Discovery & Inventory > Settings****.*

`ExcludeDirectory` has two closely-related effects:

1. It excludes specified folder(s) from the path(s) for file inventory scanning declared in `IncludeDirectory`. If `Recurse` is `True` , then all subfolders of the excluded folder are also excluded. Notice that in this case, the folder must *first* be specified in `IncludeDirectory`, or else `ExcludeDirectory` has no effect on the paths scanned for regular file inventory.
2. It excludes specified folder(s) from the system-wide scan undertaken when PerformOracleFMWScan is true. In this case, there is no preliminary requirement to specify any inclusions, as Oracle requires the system-wide scan.

The combination of these two purposes means that you should use `ExcludeDirectory` with care, being sure not to accidentally exclude a path for reporting to Oracle when controlling the paths to search for normal file evidence.

Tip: *A folder may still be scanned, but as part of the search for ISO tag files, if it is included through EmbedFileContentDirectory and not also excluded by ExcludeEmbedFileContentDirectory. Even in this case, no normal file inventory is returned from a path identified in* `ExcludeDirectory`*. However, note that by default, the same values set in the web interface are copied both to the preferences for file evidence and to the preferences for ISO-standard SWID tags (the "*EmbedFileContentDirectory" pair). This means that normally the same paths are searched (or excluded) for both file evidence and SWID tags.*

This preference can accept multiple values in a list separated by either commas or semi-colons.

Tip: *Comma separators can be used freely. If you use semi-colon separators, it is mandatory on UNIX-like platforms to enclose the list of values in either single or double ASCII quotation marks. (Be careful that any copy/paste doesn't switch to 'smart quotes', and use only plain ASCII quotation marks.)*

The value can symbolically refer to another preference by enclosing its name thus: `$(`*`preferenceName`*`)`. References can contain further references.

If a folder is identified in both the `ExcludeDirectory` and `IncludeDirectory` preferences, it is excluded. Exclusions always override inclusions.

Note: *Inclusions and exclusions can cover folders (and optionally their sub-folders), file name extensions, specific file names, and specific MD5 digest values. To resolve conflicting specifications, the specifications of folders provides a data set to which the following specifications are applied as filters, prioritized from lowest to highest as:*

- *File extension*
- *File name*
- *MD5 value.*

For example, if file extension `exe` *is included, filename* `xcopy.exe` *excluded, and MD5 value* `123456...` *(the MD5 for* `xcopy.exe`*) is included, then the inventory agent includes all files with extension* `exe` *except for all versions of* `xcopy.exe` *that do not have an MD5 value* `123456...`*.*

Values

Values / range	Valid folders

Default value	Default behavior on Windows platforms:
	`$(WinDirectory)`
	 Note: *This default prevents inventory collection of potentially millions of file evidence records from (by default) `C:\Windows` and is subfolders. The default is not visible in the `[Registry]` but can be over-ridden in the command line as shown below.*
	Default behavior on Linux platforms:
	`/var/lib/docker`
	 Note: *A folder may still be scanned, but as part of the search for general file inventory, if it is included through IncludeDirectory and not also excluded by ExcludeDirectory. Even in this case, no ISO tag files are returned from a path identified in `ExcludeEmbedFileContentDirectory`.*

Example value	This example adds another folder to the current default behavior on Windows platforms:
	`$(WinDirectory);C:\Temp`
	Another way to add an exclusion to the current default behavior is:
	`$(ExcludeDirectory);C:\Temp`
	If you specify a path in other ways that do not also include the existing default, you over-ride that default, and re-introduce the large volume of file evidence records possible in the Windows folder. For example, this parameter on Windows platforms would include inventory of the `$(WinDirectory)` folder and its subfolders:
	`C:\Temp`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o ExcludeDirectory=C:\Temp`

Registry

Installed by	Manual configuration (otherwise predefined within the tracker).

Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

ExcludedMSIs

Command line | Registry

For Microsoft Windows devices only, `ExcludedMSIs` prevents the recording of application usage data for specified native package format (MSI) applications. The applications are excluded when their MSI product code GUID is listed in this preference. Once excluded, no application usage data is recorded for the applications installed from the listed MSI packages. This preference can accept multiple comma-separated values.

Values

Values / range	Valid application GUIDs, enclosed in curly braces, and (for multiples) comma-separated.
Default value	The product code GUID for the FlexNet inventory agent.
Example values	`{00000409-78E1-11D2-B60F-006097C998E7}`

Command line

Tool	Application usage component (`mgsusageag`)
Example	No command line support.

Registry

Installed by	Installation of the FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

ExcludeEmbedFileContentDirectory

Command line | Registry

`ExcludeEmbedFileContentDirectory` identifies a subpath within the current search target (declared in EmbedFileContentDirectory) that must be excluded from scanning for ISO tag files.

This preference may be set by downloaded policy to reflect the settings in the web interface for FlexNet Manager Suite at **Discovery & Inventory > Settings**.

***Tip:** A folder may still be scanned, but as part of the search for general file inventory, if it is included through*

IncludeDirectory and not also excluded by ExcludeDirectory. Even in this case, no ISO tag files are returned from a path identified in `ExcludeEmbedFileContentDirectory`.

Values

Values / range	One (or more) folder(s) to be excluded from the search path scanned for ISO tag files to embed in the uploaded inventory (`.ndi`) file. Multiple paths should be separated by the semi-colon character.
Default value	Default behavior on Linux platforms: `/var/lib/docker` ***Note:*** *A folder may still be scanned, but as part of the search for general file inventory, if it is included through IncludeDirectory and not also excluded by ExcludeDirectory. Even in this case, no ISO tag files are returned from a path identified in `ExcludeEmbedFileContentDirectory`.* There is no default behavior on other platforms.
Example values	`"C:\Program Files (x86)\Adobe"` This example excludes Adobe applications from checking for ISO tag files (perhaps to save scanning folders containing non-standard file extensions — see EmbedFileContentExtension).

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o ExcludeEmbedFileContentDirectory="C:\Program Files (x86)\Adobe"`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

ExcludeExtension

Command line | Registry

`ExcludeExtension` excludes files with the specified extension from inventory, or excludes all files if set to the value * (asterisk). This filter is applied to the files within a folder included in inventory. This preference can accept multiple values, separated by commas or semicolons.

Tip: *Comma separators can be used freely. If you use semi-colon separators, it is mandatory on UNIX-like platforms to enclose the list of values in either single or double ASCII quotation marks. (Be careful that any copy/paste doesn't switch to 'smart quotes', and use only plain ASCII quotation marks.)*

You can specify any valid file extensions (no leading dot required). Be aware that including `exe` in this list prevents tracking of executable files on Windows platforms.

Note: *Inclusions and exclusions can cover folders (and optionally their sub-folders), file name extensions, specific file names, and specific MD5 digest values. To resolve conflicting specifications, the specifications of folders provides a data set to which the following specifications are applied as filters, prioritized from lowest to highest as:*

- *File extension*
- *File name*
- *MD5 value.*

For example, if file extension `exe` is included, filename `xcopy.exe` excluded, and MD5 value `123456...` (the MD5 for `xcopy.exe`) is included, then the inventory agent includes all files with extension `exe` except for all versions of `xcopy.exe` that do not have an MD5 value `123456...`.

Values

Values / range	File extensions (no period required)
Default value	(No default.)
Example value	`DLL`

Command line

Tool	Inventory agent
Example	`-o ExcludeExtension=dll`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

ExcludeFile

Command line | Registry

`ExcludeFile` excludes a specific file from inventory collection. This filter is applied to files within a folder included in inventory. This preference can accept multiple values, separated by commas or semi-colons.

Tip: *Comma separators can be used freely. If you use semi-colon separators, it is mandatory on UNIX-like platforms to enclose the list of values in either single or double ASCII quotation marks. (Be careful that any copy/paste doesn't switch to 'smart quotes', and use only plain ASCII quotation marks.)*

Note: *Inclusions and exclusions can cover folders (and optionally their sub-folders), file name extensions, specific file names, and specific MD5 digest values. To resolve conflicting specifications, the specifications of folders provides a data set to which the following specifications are applied as filters, prioritized from lowest to highest as:*

- *File extension*
- *File name*
- *MD5 value.*

For example, if file extension `exe` is included, filename `xcopy.exe` excluded, and MD5 value `123456...` (the MD5 for `xcopy.exe`) is included, then the inventory agent includes all files with extension `exe` except for all versions of `xcopy.exe` that do not have an MD5 value `123456...`.

Values

Values / range	Valid file names.
Default value	(No default.)
Example value	`myfile.txt`

Command line

Tool	Inventory agent
Example	`-o ExcludeFile=myfile.txt`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

ExcludeFileSystemType

Command line | Registry

For UNIX-like devices, `ExcludeFileSystemType` allows you to prevent inventory collection from specific types of files systems that may otherwise be included. (This preference is not applicable to Microsoft Windows.)

This file system blocklist is complemented by an safelist in `IncludeFileSystemType`. Note that if a file system type is specified in both lists, the exclude has priority.

Keep in mind the potential interaction between the following preferences:

- `IncludeDirectory`
- `IncludeNetworkDrives`
- `IncludeFileSystemType` and `ExcludeFileSystemType` (on UNIX-like systems only).

Of these, `IncludeDirectory` has lowest priority, and priority increases down the list. This means that, on UNIX-like systems, `IncludeFileSystemType` (the highest priority) can be an exception to the general rule that "exclude overrides include". The following table illustrates cases where the other settings may over-ride the `IncludeDirectory` preference, where that specifies a network drive. The first three cases fit the general rule, but the fourth is a special case:

...FileSystemType	IncludeNetworkDrives	IncludeDirectory	Result
Not specified (including Microsoft Windows)	`False` (this setting operates like an "exclude")	A network share (or subdirectory thereof) on any file system type.	No inventory returned.
Not specified (including Microsoft Windows)	`True`	A network share (or subdirectory thereof) on any file system type.	Software inventory from the specified directory.
`ExcludeFileSystemType=XXX`	Either `True` or `False` (here, a `True` setting is over-ridden by the exclude)	A network share (or subdirectory thereof) on file system type `XXX` (here, the setting is over-ridden by the exclude).	No inventory returned.
`IncludeFileSystemType=XXX`	Either `True` or `False` (here, a `False` setting is over-ridden, because the file system type has higher precedence)	A network share (or subdirectory thereof) on file system type `XXX`.	Software inventory from the specified directory.

Tip: *`IncludeDirectory` is not the only preference that causes folders to be scanned. See also EmbedFileContentDirectory.*

Values

Values / range	Comma-separated list of standard file system type names, as recognized by the UNIX `mount` command. Either omit white space, or enclose the list in double quotation marks.
Default value	No `[Registry]` default value, and no default behavior. (Equivalent to a default value of "".)
Example value	`zfs` This value excludes inventory collection from the `zfs` file system. This modifies the default behavior (for more details, see IncludeFileSystemType).

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o ExcludeFileSystemType=zfs`

Registry

Installed by	Manual configuration in `/var/opt/ managesoft/etc/config.ini` (see [Registry] Explained).
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

ExcludeLocalScriptRule

Command line | Registry

Tip: *This preference requires that the up-to-date `InventorySettings.xml` file is either:*

- *Co-located with the FlexNet Inventory Scanner (or the scanner-like `ndtrack.sh` on UNIX-like platforms)*
- *Correctly installed with the fully-installed FlexNet inventory agent.*

Unless this condition is met, this preference is ignored.

By default when the above condition is met, the tracker (`ndtrack` executable) will perform all local scripting actions specified in `InventorySettings.xml`. For example, this file includes enhanced inventory abilities related to Oracle databases and the hardware they run on, and Microsoft Exchange (although the functionality available is subject to the products you have licensed within FlexNet Manager Suite). When the `ExcludeLocalScriptRule` preference is included, any local script rules that are included as parameters in the list will *not* be performed (while any rules not identified will continue to run). This means that you should use this preference only when you want to exclude specific scripts because they are not required, or are affecting performance, or are outside your security policies.

Tip: *As an alternative to the* `ExcludeLocalScriptRule` *preference, you can use the* `IncludeLocalScriptRule` *preference (which runs only the named scripts and implicitly excludes all others). While it is not recommended that you use both preferences at the same time, if both are used,* `ExcludeLocalScriptRule` *takes precedence.*

Tip: *Collection of Oracle inventory may be affected by any of the following preferences:*

- *PerformLocalScripting*
- *PerformOracleInventory*
- *PerformOracleListenerScan*
- *ExcludeLocalScriptRule*
- *IncludeLocalScriptRule*.

Values

Values / range	Valid rule name. For reference, valid rule names can be found in `InventorySettings.xml` by searching for the `Id` attribute of the `RecognitionRule` XML element.
Default value	There is no default. You must specify a rule name as a value. When neither of `ExcludeLocalScriptRule` or `IncludeLocalScriptRule` is specified, the default is to execute all the scripts in `InventorySettings.xml` (assuming `PerformLocalScripting` is true).

Example values	`OracleCPURule`
	This rule controls the collection of Oracle hardware inventory.
	`OracleRule`
	This rule controls the collection of Oracle Database inventory.

Command line

Tool	`ndtrack`

Example	`-o ExcludeLocalScriptRule="OracleCPURule"`
	To exclude more than one rule, use a comma-delimited list:
	`-o ExcludeLocalScriptRule="OracleCPURule,OracleRule,`*`AdditionalRule`*`"`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\Managesoft\Tracker\CurrentVersion`

ExcludeMD5

Command line | Registry

For files within a folder included in inventory, the tracker performs an MD5 checksum, and excludes any files from the inventory that have an MD5 value equal to any value stored in `ExcludeMD5`. This preference can accept multiple values, separated by commas or semicolons.

Tip: *Comma separators can be used freely. If you use semi-colon separators, it is mandatory on UNIX-like platforms to enclose the list of values in either single or double ASCII quotation marks. (Be careful that any copy/paste doesn't switch to 'smart quotes', and use only plain ASCII quotation marks.)*

Note: *Inclusions and exclusions can cover folders (and optionally their sub-folders), file name extensions, specific file names, and specific MD5 digest values. To resolve conflicting specifications, the specifications of folders provides a data set to which the following specifications are applied as filters, prioritized from lowest to highest as:*

- *File extension*
- *File name*
- *MD5 value.*

For example, if file extension `exe` is included, filename `xcopy.exe` excluded, and MD5 value `123456...` (the MD5 for `xcopy.exe`) is included, then the inventory agent includes all files with extension `exe` except for all versions of `xcopy.exe` that do not have an MD5 value `123456...`.

Values

Values / range	Valid MD5 values
Default value	(No default.)
Example value	`7d9d2440656fdb3645f6734465678c60`

Command line

Tool	Inventory agent
Example	`-o ExcludeMD5=7d9d2440656fdb3645f6734465678c60`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

ExecutablePath

Registry

`ExecutablePath` specifies the file path and (normally) the executable file name to be monitored for application usage. This preference is required only when the manual mapper is used to identify files to monitor (and is otherwise ignored). This registry key must be created manually, within a node for the chosen application that has also been inserted manually (and shown below as *`Application node`*).

The specification of the file path may include simple regular expressions, provided that this same *`Application node`* also includes the value `Regex=True` (see third example below). For details of typical regex patterns, see Regex.

Note: *The value for `ExecutablePath` may be either:*

- *An absolute path and executable file name (see first example below).*
- *An absolute path to a folder, without specifying the executable file name (see second example below). In this case, every executable found in the folder is monitored. On Windows devices, you may not use this format for Windows system folders – for example, folder names (alone, without executables) like `C:\Windows` and `C:\Windows\System32` are ignored. To track usage of an executable in such paths, be sure to include the executable file name.*

Values

Values / range	Valid text file path specification.
Default value	No default value.

Example values	`C:\WINNT\System32\sol.exe`
	`C:\Program Files\LAPS`
	`.*\sol[.]exe`

Registry

Installed by	Manual configuration only.
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion\Manual Mapper\`*`Application node`*

GenerateMD5

Command line | Registry

`GenerateMD5` specifies whether or not to calculate the MD5 digest of any file being considered by the tracker, and to include it with stored inventory data. Since MD5 digests are not used to identify versions of inventoried files, set this preference to `True` only when it is needed to support an `IncludeMD5` or `ExcludeMD5` preference. Be aware that calculating MD5 digests will degrade performance where many files are being tracked.

Values

Values / range	Boolean (`True` or `False`).
Default value	`False`
Example value	`True`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o GenerateMD5=True`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

Hardware

Command line | Registry

`Hardware` allows you to track hardware either using Windows Management Instrumentation (WMI) or native APIs. (If WMI is available, by default it is used for tracking — see WMI.) When set to `True`, `Hardware` allows the tracking of hardware inventory. When set to `False`, the inventory tool does not track hardware inventory.

Values

Values / range	Boolean (`True` or `False`)

Default value	`True`
Example value	`False`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o Hardware=False`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

HardwareChangesClassPropertyBlacklist

Command line | Registry

`HardwareChangesClassPropertyBlacklist` is used when FlexNet Manager Suite is in high-frequency mode for calculating subcapacity consumption of IBM PVU licenses.

In this mode, the FlexNet inventory agent, locally installed on each target inventory device, must check for relevant hardware changes on a set schedule (every 30 minutes is the requirement from IBM). In order to reduce network traffic, the installed FlexNet inventory agent (using its default command line for this case) only creates and uploads an inventory (`.ndi`) file when there is a relevant hardware change.

The purpose of `HardwareChangesClassPropertyBlacklist` is to specify those attributes of a hardware device that may be ignored. Value changes in these listed properties do *not* cause generation, compression, and upload of an inventory file.

This option has no effect if `OnlyGenerateIfHardwareChanged` is false (see OnlyGenerateIfHardwareChanged).

Values

Values / range	A string value made up of semi-colon-separated WMI classes. Each entry may be a simple class name, or a class name with trailing class property/properties (using a dot separator).

Default value	`Win32_OperatingSystem.FreePhysicalMemory.FreeVirtualMemory;LastBootUpTime.InstallDate;`
	`Win32_Processor.CurrentClockSpeedNonWMI.CurrentClockSpeed;`
	`Win32_LogicalDisk.FreeSpace;`
	`SoftwareLicensingProduct;`
	`MGS_OperatingSystem.LastBootUpTime.FreePhysicalMemory.FreeVirtualMemory.FreeSpaceInPagingFiles;`
	`MGS_AgentEvent;`
	`Win32_NetworkAdapter;`
	`Win32_NetworkAdapterConfiguration;`
	`MGS_NetworkAdapterConfiguration;`
	`MGS_Processor.CurrentClockSpeed.MaxClockSpeed`
	
	This value applies when nothing in specified in either [Registry] or command line.

Example values	`Win32_LogicalDisk.FreeSpace` Any declared value completely replaces the default behavior. This example has the effect of 'turning on' the five other values in the above default, so that changes in those other properties now trigger an inventory upload; and only changes in free disk space are ignored.

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o HardwareChangesClassPropertyBlacklist="Win32_LogicalDisk.FreeSpace"`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

HighestPriority

Registry

`HighestPriority` specifies the highest upload/download priority that can be assigned to an inventory beacon. The lower the number, the higher the priority.

When assigning priorities, an inventory device normalizes the calculated priority to fit within the range identified by `HighestPriority` and `LowestPriority`. The highest priority is commonly set to 1.

Values

Values / range	Recommended 1-100 (but can extend from -2^{31} to 2^{31}).
Default value	No default in registry; default behavior uses 10.
Example values	`10`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\NetSelector\CurrentVersion`

Host

Registry

`Host` identifies the inventory beacon on which the reporting location (uploads) or distribution location (downloads) is hosted.

The value may, as appropriate, be either the simple host name (within a single domain) or the fully qualified domain name (FQDN) of the inventory beacon. Of course, name resolution depends on correctly-maintained domain name servers (DNS). Within network segments that have IPv4 transport protocol available, an IP address (in IPv4 format) may be supplied when necessary (in segments supporting only the IPv6 addressing standard, use of the host's name is mandatory, and an IPv6 address may not be used).

Values

Values / range	Valid host name
Default value	(No default.)
Example values	`server.domain.com`

Registry

Installed by	Failover list for inventory beacons, or manual configuration

Computer preference	For uploads:
	`[Registry]\ManageSoft\Common\UploadSettings\<reporting_location>`
	For downloads:
	`[Registry]\ManageSoft\Common\DownloadSettings\<distribution_location>`
	For trusted locations:
	`[Registry]\ManageSoft\Launcher\CurrentVersion\TrustedLocations\<serverkey>`
	For excluded locations:
	`[Registry]\ManageSoft\Launcher\CurrentVersion\ExcludedLocations\<serverkey>`

http_proxy

Command line | Registry

`http_proxy` gives the proxy settings for the installation component (`ndlaunch`) when using the HTTP protocol. See also no_proxy.

Values

Values / range	Any valid URL
Default value	Not to use a proxy.
Example values	`tmnis.com;tmnis.com.de`

Command line

Tool	Installation component (`ndlaunch`)
Example	`-o http_proxy=tmnis.com;tmnis.com.de`

Registry

Installed by	Installation of FlexNet inventory agent on a managed device (computer preference)

Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`

https_proxy

Command line | Registry

`https_proxy` gives the proxy settings for the installation component (`ndlaunch`) when using the HTTPS protocol. See also no_proxy.

Values

Values / range	Any valid URL
Default value	Not to use a proxy.
Example values	`tmnis.com;tmnis.com.de`

Command line

Tool	Installation component (`ndlaunch`)
Example	`-o https_proxy=tmnis.com;tmnis.com.de`

Registry

Installed by	Installation of FlexNet inventory agent on a managed device (computer preference)
Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`

IBMDB2CommandTimeoutSeconds

Command line | Registry

`IBMDB2CommandTimeoutSeconds` specifies the number of seconds that the locally-installed FlexNet inventory agent waits for a response to IBM Db2 commands that it issues while collecting inventory for the IBM Db2 Database and its optional add-ons (for details, see PerformIBMDB2Inventory). The timeout prevents the tracker hanging while waiting for a response from the database, so that other inventory collection tasks can proceed. However, be aware that if the `db2licm` command does not respond before the timeout, the related inventory gathered from this device is incomplete, and this may affect the accuracy of the **IBM Db2 Database and Add-Ons** report.

Values

Values / range	A positive integer in the range 0-2147483647. Negative or non-integer values are restored to zero. The special value of zero means that there is no timeout, and commands (and therefore all inventory gathering) will hang if there is no response from IBM Db2. Useful values are in the range of 1 to (say) 120 seconds.
Default value	`120` This default (in seconds) is automatically applied when there is no value available in the [Registry] or command line.
Example values	`10`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o IBMDB2CommandTimeoutSeconds=30`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

IncludeDirectory

Command line | Registry

Important: *This preference cannot be set manually in the registry for Microsoft Windows, nor in the `config.ini` file for UNIX-like devices. Any value set manually in the [pseudo-]registry is overwritten each time that device policy is downloaded from an inventory beacon. Preferences for the installed and policy-managed FlexNet inventory agent follow your settings in the web interface for FlexNet Manager Suite at* ***Discovery & Inventory > Settings*** *(see the discussion of default values below). You may override the saved values by using a custom command-line, as shown below.*

`IncludeDirectory` targets a specified folder for scanning for files to include in inventory. If `Recurse` is `True`, then all subfolders are also included.

Important: *The FlexNet inventory agent never follows symbolic links (on UNIX-like platforms) or NTFS junction points (on Microsoft Windows). This restriction cannot be changed. On some UNIX servers, for example, `/bin` is a symbolic*

link to `/usr/bin`. On such a system, if you specified `IncludeDirectory=/bin`, no inventory would be collected, because the symbolic link cannot be followed. Changing the preference on such a system to `IncludeDirectory=/usr/bin` solves the problem, since there is now no intermediate symbolic link. On UNIX-like systems, check for symbolic links with the `ls -l` command.

For UNIX-like platforms, a setting of "`/`" scans the entire file system. For Microsoft Windows platforms, when the value of this entry is set to "`\`", it means "include all drives", with the exception of `$(WindowsFolder)` and its subfolders. (To change this default exclusion, see ExcludeDirectory.) Notice that "include all drives" means all fixed drives *and* all local drives (such as USB sticks or USB-connected external drives), but does *not* include network shares.

Restriction: *`IncludeDirectory` sets the folder(s) that the FlexNet inventory agent is to scan for software inventory, and in particular for file evidence. However, this setting does not control scanning for Oracle Fusion Middleware (for which see PerformOracleFMWScan). As required by Oracle, a scan for Oracle Fusion Middleware must cover the entire file system; but scanning is optimized so that each folder is scanned only once. There are two possible outputs from this scan:*

- *For folders declared within `IncludeDirectory`, standard file evidence is uploaded (as always)*
- *For folders where Oracle Fusion Middleware is discovered, the specific data required by Oracle is structured, zipped into an archive, and included as a blob of binary data in the uploaded `.ndi` file.*

For the collection of regular file evidence, `IncludeDirectory` can accept multiple values, separated by commas or semi-colons.

Tip: *Comma separators can be used freely. If you use semi-colon separators, it is mandatory on UNIX-like platforms to enclose the list of values in either single or double ASCII quotation marks. (Be careful that any copy/paste doesn't switch to 'smart quotes', and use only plain ASCII quotation marks.)*

If a folder is identified in both the `ExcludeDirectory` and `IncludeDirectory` preferences, it is excluded. Exclusions (of the same thing) always override inclusions.

Note: *Inclusions and exclusions can cover folders (and optionally their sub-folders), file name extensions, specific file names, and specific MD5 digest values. To resolve conflicting specifications, the specifications of folders provides a data set to which the following specifications are applied as filters, prioritized from lowest to highest as:*

- *File extension*
- *File name*
- *MD5 value.*

For example, if file extension `exe` is included, filename `xcopy.exe` excluded, and MD5 value `123456...` (the MD5 for `xcopy.exe`) is included, then the inventory agent includes all files with extension `exe` except for all versions of `xcopy.exe` that do not have an MD5 value `123456...`.

Keep in mind the potential interaction between the following preferences:

- `IncludeDirectory`
- `IncludeNetworkDrives`
- `IncludeFileSystemType` and `ExcludeFileSystemType` (on UNIX-like systems only).

Of these, `IncludeDirectory` has lowest priority, and priority increases down the list. This means that, on UNIX-like systems, `IncludeFileSystemType` (the highest priority) can be an exception to the general rule that "exclude overrides include". The following table illustrates cases where the other settings may over-ride the `IncludeDirectory` preference, where that specifies a network drive. The first three cases fit the general rule, but the fourth is a special case:

...FileSystemType	IncludeNetworkDrives	IncludeDirectory	Result
Not specified (including Microsoft Windows)	`False` (this setting operates like an "exclude")	A network share (or subdirectory thereof) on any file system type.	No inventory returned.
Not specified (including Microsoft Windows)	`True`	A network share (or subdirectory thereof) on any file system type.	Software inventory from the specified directory.
`ExcludeFileSystemType=XXX`	Either `True` or `False` (here, a `True` setting is over-ridden by the exclude)	A network share (or subdirectory thereof) on file system type `XXX` (here, the setting is over-ridden by the exclude).	No inventory returned.
`IncludeFileSystemType=XXX`	Either `True` or `False` (here, a `False` setting is over-ridden, because the file system type has higher precedence)	A network share (or subdirectory thereof) on file system type `XXX`.	Software inventory from the specified directory.

Tip: *`IncludeDirectory` is not the only preference that causes folders to be scanned. See also EmbedFileContentDirectory.*

Values

Values / range	Valid folder(s)

Default value	Different defaults for different cases: • For the full FlexNet inventory agent driven by policy downloaded from an inventory beacon, the value reflects the current settings displayed in the web interface (navigate to **Discovery & Inventory > Settings > File inventory**). The standard setting on that page triggers scanning of the entire file system. • For the lightweight FlexNet Inventory Scanner case (including `ndtrack.sh` on UNIX-like platforms), the default is blank. This means that *no files are included in inventory gathering by default* when you use the FlexNet Inventory Scanner. Reported inventory then relies entirely on installer evidence. If you wish the FlexNet Inventory Scanner to collect file evidence for any reason, it is mandatory to set an appropriate value for `IncludeDirectory`.
Example value	`C:\Program Files`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o IncludeDirectory=C:\Temp`

Registry

Installed by	Installation of FlexNet inventory agent on a managed device (computer preference), or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

IncludeExecutables

Command line | Registry

`IncludeExecutables` extends the meaning of 'executable files' across platforms. If `IncludeDirectory` is blank (its default), this setting has no effect. However, when `IncludeDirectory` has any value:

- A setting of `False` means that only files with a file extension of `.exe` are included in executable files inventory (normally, this means only Windows executables)
- A setting of `True` means that (for files not already matched by other settings) the tracker will report:
  - On Windows, files with an `exe` filename extension
  - On UNIX-like platforms, files with no filename extension and an execute bit set (in any of local, group, or universal scope in the file permission bits).

Values

Values / range	Boolean (`True` or `False`)
Default value	`True`
Example value	`False`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o IncludeExecutables=True`

Registry

Installed by	Installation of FlexNet inventory agent on a device, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

IncludeExtension

Command line | Registry

`IncludeExtension` includes files with the specified extension, or includes all files if this preference is set to the value `*`. This filter is applied within one or more folders included in inventory (see IncludeDirectory). This preference can accept multiple values, separated by commas or semi-colons.

Tip: *Comma separators can be used freely. If you use semi-colon separators, it is mandatory on UNIX-like platforms to enclose the list of values in either single or double ASCII quotation marks. (Be careful that any copy/paste doesn't switch to 'smart quotes', and use only plain ASCII quotation marks.)*

The dot separator for file extensions is not required.

Note: *Inclusions and exclusions can cover folders (and optionally their sub-folders), file name extensions, specific file names, and specific MD5 digest values. To resolve conflicting specifications, the specifications of folders provides a data set to which the following specifications are applied as filters, prioritized from lowest to highest as:*

- *File extension*
- *File name*
- *MD5 value.*

For example, if file extension `exe` is included, filename `xcopy.exe` excluded, and MD5 value `123456...` (the MD5 for `xcopy.exe`) is included, then the inventory agent includes all files with extension `exe` except for all versions of `xcopy.exe` that do not have an MD5 value `123456...`.

Values

Values / range	Any file extension
Default value	`"sys;sys2;swtag;cmptag;lax;swidtag;sig;exe"` ***Tip:*** *The `exe` extension is applicable only to Windows platforms.*
Example value	`bat`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o IncludeExtension=bat`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

IncludeFile

Command line | Registry

`IncludeFile` searches for the specific file to report in inventory; but keep in mind that the search is constrained to folders that are specified as included in inventory. This preference can accept multiple values, separated with commas or semi-colons.

Tip: *Comma separators can be used freely. If you use semi-colon separators, it is mandatory on UNIX-like platforms to enclose the list of values in either single or double ASCII quotation marks. (Be careful that any copy/paste doesn't switch to 'smart quotes', and use only plain ASCII quotation marks.)*

Note: *Inclusions and exclusions can cover folders (and optionally their sub-folders), file name extensions, specific file names, and specific MD5 digest values. To resolve conflicting specifications, the specifications of folders provides a*

data set to which the following specifications are applied as filters, prioritized from lowest to highest as:

- *File extension*
- *File name*
- *MD5 value.*

For example, if file extension `exe` is included, filename `xcopy.exe` excluded, and MD5 value `123456...` (the MD5 for `xcopy.exe`) is included, then the inventory agent includes all files with extension `exe` except for all versions of `xcopy.exe` that do not have an MD5 value `123456...`.

Values

Values / range	Valid file
Default value	(No default.)
Example value	`xcopy.exe`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o IncludeFile=myfile.txt`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

IncludeFileSystemType

Command line | Registry

For UNIX-like devices, `IncludeFileSystemType` allows you to include specific types of files systems that may otherwise be excluded. (This preference is not applicable to Microsoft Windows.)

The `ufs`, `zfs`, and `lofs` file systems use virtual device nodes (within the operating system) rather than listing their drives as physical `/dev/...` devices. This makes it more difficult to determine whether they are local or remote file systems, and they would therefore logically be excluded when `IncludeNetworkDrives=False` (the default). When you have these file systems running locally on a UNIX-like device, this setting allows you to exclude network drives and still allow inventory collection from the local file system by nominating the file system type(s).

This file system safelist is complemented by a blocklist in `ExcludeFileSystemType`. Note that if a file system type is

specified in both lists, the exclude has priority.

Keep in mind the potential interaction between the following preferences:

- `IncludeDirectory`
- `IncludeNetworkDrives`
- `IncludeFileSystemType` and `ExcludeFileSystemType` (on UNIX-like systems only).

Of these, `IncludeDirectory` has lowest priority, and priority increases down the list. This means that, on UNIX-like systems, `IncludeFileSystemType` (the highest priority) can be an exception to the general rule that "exclude overrides include". The following table illustrates cases where the other settings may over-ride the `IncludeDirectory` preference, where that specifies a network drive. The first three cases fit the general rule, but the fourth is a special case:

...FileSystemType	IncludeNetworkDrives	IncludeDirectory	Result
Not specified (including Microsoft Windows)	`False` (this setting operates like an "exclude")	A network share (or subdirectory thereof) on any file system type.	No inventory returned.
Not specified (including Microsoft Windows)	`True`	A network share (or subdirectory thereof) on any file system type.	Software inventory from the specified directory.
`ExcludeFileSystemType=XXX`	Either `True` or `False` (here, a `True` setting is over-ridden by the exclude)	A network share (or subdirectory thereof) on file system type `XXX` (here, the setting is over-ridden by the exclude).	No inventory returned.
`IncludeFileSystemType=XXX`	Either `True` or `False` (here, a `False` setting is over-ridden, because the file system type has higher precedence)	A network share (or subdirectory thereof) on file system type `XXX`.	Software inventory from the specified directory.

Tip: *`IncludeDirectory` is not the only preference that causes folders to be scanned. See also EmbedFileContentDirectory.*

Values

Values / range	Comma-separated list of standard file system type names, as recognized by the UNIX `mount` command. Either omit white space, or enclose the list in double quotation marks.

Default value	No `[Registry]` default value. If no value is specified in the registry or command line, the default behavior is `ufs,zfs,lofs`.
Example value	`ufs,lofs` Specifying any value for this preference overrides (replaces) the default behavior. This example value excludes the `zfs` file system (otherwise included in the default), which exclusion may be required if this is in fact a remote file system.

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o IncludeFileSystemType=ufs,lofs`

Registry

Installed by	Manual configuration in `/var/opt/ managesoft/etc/config.ini` (see [Registry] Explained).
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

IncludeLocalScriptRule

Command line | Registry

Tip: *This preference requires that the up-to-date `InventorySettings.xml` file is either:*

- *Co-located with the FlexNet Inventory Scanner (or the scanner-like `ndtrack.sh` on UNIX-like platforms)*
- *Correctly installed with the fully-installed FlexNet inventory agent.*

Unless this condition is met, this preference is ignored.

By default when the above condition is met, the tracker (`ndtrack` executable) will perform all local scripting actions specified in `InventorySettings.xml`. For example, this file includes enhanced inventory abilities related to Oracle databases and the hardware they run on, and Microsoft Exchange (although the functionality available is subject to the products you have licensed within FlexNet Manager Suite). When the `IncludeLocalScriptRule` preference is included, only local script rules that are included as parameters in the list will be performed (all others are implicitly excluded). This means that you should use this preference only when you want to limit the number of local scripts running on an inventory device.

Tip: *As an alternative, the `ExcludeLocalScriptRule` preference lists specific rules to exclude, leaving all others running. If you use both preferences, `ExcludeLocalScriptRule` has priority.*

Tip: *Collection of Oracle inventory may be affected by any of the following preferences:*

- *PerformLocalScripting*
- *PerformOracleInventory*
- *PerformOracleListenerScan*
- *ExcludeLocalScriptRule*
- *IncludeLocalScriptRule*.

Values

Values / range	Valid rule name. For reference, valid rule names can be found in `InventorySettings.xml` by searching for the `Id` attribute of the `RecognitionRule` XML element.
Default value	There is no default. You must specify a rule name as a value. When neither of `IncludeLocalScriptRule` or `ExcludeLocalScriptRule` is specified, the default is to execute all the scripts in `InventorySettings.xml` (assuming `PerformLocalScripting` is true).

Example values	`OracleCPURule`
	This rule controls the collection of Oracle hardware inventory.
	`OracleRule`
	This rule controls the collection of Oracle Database inventory.

Command line

Tool	`ndtrack`

Example	`-o IncludeLocalScriptRule="OracleCPURule"`
	To include more than one rule, use a comma-delimited list:
	`-o IncludeLocalScriptRule="OracleCPURule,AdditionalRule"`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\Managesoft\Tracker\CurrentVersion`

IncludeMachineInventory

Command line | Registry

`IncludeMachineInventory` If `True`, the tracker performs a computer inventory including hardware and all user packages.

Values

Values / range	Boolean (`True` or `False`).
Default value	Different default for different platforms: • On Microsoft Windows, for both the full FlexNet inventory agent and the lightweight FlexNet Inventory Scanner: `True` if running as `LocalSystem` or running a machine inventory on the command line. • On UNIX-like platforms, always `True`. This value cannot be over-ridden on UNIX-like systems.
Example value	`False`

Command line

Tool	Inventory component (`ndtrack`)
Example	(Microsoft Windows only): `-o IncludeMachineInventory=False`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

IncludeMD5

Command line | Registry

`IncludeMD5` includes in the inventory report any files having the specific MD5 digest. This filter is applied to files within a folder included in inventory. This preference can accept multiple values, separated with commas or semi-colons.

Tip: *Comma separators can be used freely. If you use semi-colon separators, it is mandatory on UNIX-like platforms to enclose the list of values in either single or double ASCII quotation marks. (Be careful that any copy/paste doesn't switch to 'smart quotes', and use only plain ASCII quotation marks.)*

Note: *Inclusions and exclusions can cover folders (and optionally their sub-folders), file name extensions, specific file names, and specific MD5 digest values. To resolve conflicting specifications, the specifications of folders provides a data set to which the following specifications are applied as filters, prioritized from lowest to highest as:*

- *File extension*
- *File name*
- *MD5 value.*

For example, if file extension `exe` is included, filename `xcopy.exe` excluded, and MD5 value `123456...` (the MD5 for `xcopy.exe`) is included, then the inventory agent includes all files with extension `exe` except for all versions of `xcopy.exe` that do not have an MD5 value `123456...`.

Values

Values / range	Any valid MD5 digest
Default value	(No default.)
Example value	`7d9d2440656fdb3645f6734465678c60`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o IncludeMD5=7d9d2440656fdb3645f6734465678c60`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

IncludeNetworkDrives

Command line | Registry

`IncludeNetworkDrives` allows for the inclusion of mounted network drives in inventory. There are differences across platforms:

- On Microsoft Windows, drives are included (`True`) or excluded (`False`) as a whole.
- On UNIX-like platforms, network file systems are mounted as directories in the same way that all devices are mounted. When `Recurse` is `True`, inventory collection starts at each folder listed in `IncludeDirectory` and drills down through all child folders. If this process encounters a directory that is a mounted network drive, and `IncludeNetworkDrives` is not specified or `False`, the process goes no further down this path. However, if `IncludeNetworkDrives=True`, recursion continues. Note that this preference controls the recursion process, and does not prevent inventory scanning of a directory specified in `IncludeDirectory`, even if that is a mounted network drive.

Tip: *Use this setting with great care. Scanning mounted network drives can cause a massive increase in the amount of inventory collected, depending on the other settings used in the same execution.*

Keep in mind the potential interaction between the following preferences:

- `IncludeDirectory`
- `IncludeNetworkDrives`
- `IncludeFileSystemType` and `ExcludeFileSystemType` (on UNIX-like systems only).

Of these, `IncludeDirectory` has lowest priority, and priority increases down the list. This means that, on UNIX-like systems, `IncludeFileSystemType` (the highest priority) can be an exception to the general rule that "exclude overrides include". The following table illustrates cases where the other settings may over-ride the `IncludeDirectory` preference, where that specifies a network drive. The first three cases fit the general rule, but the fourth is a special case:

...FileSystemType	IncludeNetworkDrives	IncludeDirectory	Result
Not specified (including Microsoft Windows)	`False` (this setting operates like an "exclude")	A network share (or subdirectory thereof) on any file system type.	No inventory returned.
Not specified (including Microsoft Windows)	`True`	A network share (or subdirectory thereof) on any file system type.	Software inventory from the specified directory.
`ExcludeFileSystemType=XXX`	Either `True` or `False` (here, a `True` setting is over-ridden by the exclude)	A network share (or subdirectory thereof) on file system type `XXX` (here, the setting is over-ridden by the exclude).	No inventory returned.
`IncludeFileSystemType=XXX`	Either `True` or `False` (here, a `False` setting is over-ridden, because the file system type has higher precedence)	A network share (or subdirectory thereof) on file system type `XXX`.	Software inventory from the specified directory.

Tip: *`IncludeDirectory` is not the only preference that causes folders to be scanned. See also EmbedFileContentDirectory.*

Values

Values / range	Boolean (`True` or `False`).
Default value	No registry default value. If no value is specified in the registry or command line, the default behavior is `False`.
Example value	`True`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o IncludeNetworkDrives=True`

Registry

Installed by	Manual configuration.
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

IncludeRegistryKey

Command line | Registry

Set `IncludeRegistryKey` to instruct the FlexNet inventory agent to return the contents of the specified registry keys or values in the uploaded `.ndi` (inventory) file. (Applies to Microsoft Windows devices only, and is ignored on UNIX-like platforms.)

Note: *This setting is useful only in on-premises implementations, where it requires additional configuration of your central application server (specifically, the inventory server in larger implementations) to import this additional data and make it available in the web interfaceof FlexNet Manager Suite.*

In order to collect *all* values under a specified key, the (case sensitive) key path specified must end with a trailing backslash. If the path specified corresponds to a key (rather than a registry value) but does not end with a trailing backslash, only the `(Default)` value (if it is set) for the specified key will be collected.

Tip: *In the registry, default values are typically not set.*

For example:

- `HKLM\SOFTWARE\ManageSoft Corp\ManageSoft\` will track all values under the specified key (because it has a trailing backslash)
- `HKLM\SOFTWARE\ManageSoft Corp\ManageSoft` will only track the `(Default)` values under the specified key (where they exist).

Important: *If you choose to set this preference on a 64-bit target inventory device, add only the 32-bit registry path (as shown in the above examples), and omit the `Wow6432Node` path. From this one 32-bit setting, the FlexNet inventory agent gathers registry keys from both the 32-bit and 64-bit registry paths. In the uploaded `.ndi` file, these are differentiated with pseudo-root values of `HKLM32` and `HKLM64`, with the remainder of the paths the same (that is, `Wow6432Node` does not appear in the reported paths).*

When setting this preference, you can use:

- The * wildcard to replace a key or value (including multiple times for different key elements in a single path)
- The abbreviations `HKLM`, `HKCU`, `HKCR`, `HKU`, `HKCC` – these will be automatically expanded to appropriate values
- A semicolon or comma to separate multiple registry paths.

Tip: *Although in the FlexNet Inventory Scanner case, `ndtrack.exe` does not read preferences from the registry to modify its own behavior, it can still be instructed to collect registry keys and values to return in inventory. To modify the default value (shown below) for registry key(s) to be read by FlexNet Inventory Scanner, set the special case on the command line.*

Important: *While `IncludeRegistryKey` controls inclusion of registry values in the uploaded inventory `.ndi` file, a separate preference on the inventory server (or, in smaller implementations, the application server) must be present and set to true for this data to be saved to the FlexNet inventory database. Be precise when using your preferred registry editor to add the following settings: do not use the `Wow6432Node` path. The setting in the registry of the inventory server is:*

- *Key: `HKLM\SOFTWARE\ManageSoft Corp\ManageSoft\Reporter\CurrentVersion\Registry`*
- *Name: `SaveRegistry`*
- *Type: `REG_SZ`*
- *Value: `True`.*

*Note that the net effect of both changes is to upload the registry values and save them into the inventory database. From there, the normal full inventory import and license compliance calculations deliver the results to the compliance database, making them visible in the inventory device property sheets, on the **WMI** sub-tab of the **Evidence** tab.*

Values

Values / range	Valid registry key or value

Default value	If no value is specified, the FlexNet inventory agent (and FlexNet Inventory Scanner) use the 'Uninstall' registry entries, typically: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\` and include any keys and values thereunder in the returned inventory. ***Note:*** *If you modify the value of `IncludeRegistryKey`, be sure to include the default value as well, as this is used during collection of software inventory on the inventory device. Separate multiple paths in this preference with semicolons.*

Example values	Track all registry keys and values under `HKEY_LOCAL_MACHINE\SOFTWARE`:
	`HKLM\SOFTWARE\*\`
	Track all values (only) under `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft`:
	`HKLM\SOFTWARE\Microsoft\*`
	Use multiple wildcards:
	`HKLM\SOFTWARE\*\CurrentVersion\*\*`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o IncludeRegistryKey="HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Command Processor\"`

Registry

Installed by	Manual configuration (without which, code internals supply the default)
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

IncludeUserInventory

Command line | Registry

`IncludeUserInventory` If `True`, causes collection of user inventory.

Values

Values / range	Boolean (`True` or `False`).
Default value	`TRUE` if running as user or running a user inventory (`-t User` on the command line)
Example values	`False`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o IncludeUserInventory=False`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

InstallDefaultSchedule

Command line | Registry

When `InstallDefaultSchedule` is `True`, the policy agent invokes the installation agent to install the package specified by `DefaultSchedulePath` before applying policy. Used by automated processes for adoption of the managed device.

Values

Values / range	Boolean (`True` or `False`)
Default value	`False`
Example values	`True`

Command line

Tool	Policy component (`mgspolicy`)

Example	`-o InstallDefaultSchedule=True`

Registry

Installed by	Installation of FlexNet inventory agent
Computer preference	`[Registry]\ManageSoft\Policy Client\CurrentVersion`

InstallProfile

Command line | Registry

`InstallProfile` is set to `Public` if the package is to be installed for All Users, or `Private` if it's to be installed for an individual user.

Values

Values / range	`Public`, `Private`
Default value	(No default.)
Example values	`Public`

Command line

Tool	Installation component (`ndlaunch`)
Example	`-o InstallProfile=Public`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`

InventoryFile

Command line | Registry

`InventoryFile` identifies the name of a local copy of the inventory file.

The name may consist of system properties that can be expanded to identify a value. For example, the default value `$(UserName) on $(MachineId).ndi` expands so that the name contains the account and machine ID related to the inventory run. This format works on both Microsoft Windows and UNIX-like platforms.

Values

Values / range	`*.ndi`
Default value	`$(UserName) on $(MachineId).ndi`
Example values	`myComputer.ndi`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o InventoryFile=myfile.ndi`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

InventoryScriptsDir

Command line | Registry

Available on Windows devices only.

`InventoryScriptsDir` gives the location of your additional scripts to be run by the `ndtrack` executable immediately before inventory data is uploaded to an inventory beacon. There are no standard scripts supplied as part of FlexNet Manager Suite in this folder: it is intended for specialized scripts of your own creation and under your own control. For example, you may have used this folder to save a VB script to collect specialized inventory values. Where the preference and its specified folder exist, all scripts that exist in this location are run on each invocation of the `ndtrack` executable.

Warning: *There is no validation of the content of the scripts found in this folder. For security, it is imperative that, when your administrator creates the folder and records it in the `InventoryScriptsDir` registry key, the folder is locked down so that it is accessible only to the account running the `ndtrack` executable (by default, local `SYSTEM`), and to a trusted human administrator who has permission to install your preferred scripts in the specified folder.*

Any scripts in the folder are executed by `scripttrack.dll`, a plug-in for `ndtrack`. (This same DLL is also required for the execution of specialized inventory tasks in the `InventorySettings.xml` file.) This means that the script execution capability is supported in the following cases of FlexNet inventory gathering:

- The Adopted case
- The Zero-footprint case
- The FlexNet Inventory Scanner case
- The Agent third-party deployment case, unless you have deliberately removed `scripttrack.dll` when preparing the deployment package
- The Core deployment case, unless you have deliberately removed `scripttrack.dll` when preparing the deployment package (noting that this cannot be removed where you are relying on `InventorySettings.xml` for advanced inventory collection, as described in Core Deployment: Implementation).

As noted below, operation in any of these cases requires that the folder exists and is identified in the `InventoryScriptsDir` registry setting on the target device where the scripts must run.

Tip: *This preference is entirely independent of the `ScriptDir` preference.*

Values

Values / range	String identifying an existing directory.
Default value	`$(ProgramFiles)\ManageSoft\Tracker\Scripts\InventoryScanningOptionsInventoryScripts` This default is used when the registry key does not exist, and no value is provided on the command line. This means that this folder (if it exists) must also be locked down; or if not, its creation must be prevented, most likely by security on the parent directory.
Example values	`$(CommonAppDataFolder)\private`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o InventoryScriptDir=C:\private`

Registry

Installed by	Manual installation

Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

InventorySettingsPath

Command line | Registry

`InventorySettingsPath` identifies the directory where the FlexNet inventory agent looks for the downloaded `InventorySettings.xml` file. This file transmits all relevant settings from the central application server to the installed FlexNet inventory agent (`ndtrack`) on the local device.

The registry value is set by the Launcher (`ndlaunch`) which downloads and installs any changed package(s) with each policy update. The Launcher also installs the `InventorySettings.xml` in the same path. Thereafter, the FlexNet inventory agent recovers the path from the registry, and then opens the XML file.

Values

Values / range	`*.ndi`

Default value	On Microsoft Windows:
	`$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Tracker\InventorySettings\`
	On UNIX-like platforms:
	`/var/opt/managesoft/tracker/inventorysettings`

Example values	Do not adjust value manually. This path is included in the downloaded settings, and set automatically.

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o InventorySettingsPath="$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Tracker\InventorySettings\"`

Registry

Installed by	Launcher (`ndlaunch`)
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

InventoryType

Command line | Registry

`InventoryType` identifies the inventory type, either machine-based or user-based.

Note: *User-based inventory collection is deprecated, and included only for backward compatibility. Specifically, no policy is generated in FlexNet Manager Suite for user-based inventory collection, so that the `ndtrack` component does not normally generate any user-based inventory. Furthermore, there is no guarantee of future operation of user-based inventory, and the recommendation is to convert any legacy systems to use machine-based inventory.*

Values

Values / range	String literals `User` or `Machine`.
Default value	`User` unless the account executing `ndtrack` is `LocalSystem`, in which case the default switches to `Machine`.
Example values	`Machine`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o InventoryType=Machine`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

LauncherCommandLine

Command line | Registry

`LauncherCommandLine` specifies `ndlaunch` parameters to pass to the installation agent when applying policy information.

Values

Values / range	Valid installation agent command line parameters (see ndlaunch Command Line).
Default value	(No default.)
Example values	`-o UserInteractionLevel=Quiet` ***Note:*** *The `-o` is required as part of the value to be appended to the command line of `ndLaunch`.*

Command line

Tool	Policy component (`mgspolicy`)
Example	`-o LauncherCommandLine="-o UserInteractionLevel=Quiet"`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Policy Client\CurrentVersion`

LogFile (application usage component)

Command line | Registry

`LogFile` gives the location of the log file (on the computer device where the FlexNet inventory agent is executing) when logging is enabled for `mgsusageag` (application usage component). All actions performed by `mgsusageag` are logged to this file. If you wish to use a log file located in a folder other than the default installation log folder, specify a full pathname.

Values

Values / range	Valid log file name, including local and UNC network files.

Default value	Windows devices:
	`$(TempDirectory)\ManageSoft\usageagent.log`
	UNIX-like devices:
	`/var/opt/managesoft/log/usageagent.log`

Example values	`C:\Temp\usageagent.log` `/tmp/usageagent.log`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o Logfile=C:\temp\myusagelogfile.log`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

LogFile (installation component)

Command line | Registry

`LogFile` gives the location of the log file (on the computer device where the FlexNet inventory agent is executing) when logging is enabled for `ndlaunch` (installation component). All actions performed by `ndlaunch` are logged to this file. If you wish to use a log file located in a folder other than the default installation log folder, specify a full pathname.

Values

Values / range	Valid log file name, including local and UNC network files.
Default value	`$(TempDirectory)\ManageSoft\installation.log`
Example values	`C:\temp\MyInstallerLog.log`

Command line

Tool	Installation component (`ndlaunch`)
Example	`-o Logfile=C:\temp\mylauncherlogfile.log`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)
Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`

LogFile (inventory component)

Command line | Registry

`LogFile` gives the location and file name of the log file (on the computer device where the tracker is executing) when logging is enabled for `ndtrack`.

Values

Values / range	Valid log file name and path.

Default value	Different defaults for different cases:
	• Full FlexNet inventory agent or FlexNet Inventory Scanner on Microsoft Windows:
	`$(TempDirectory)\ManageSoft\tracker.log`
	• Full FlexNet inventory agent on UNIX-like platforms:
	`/var/opt/managesoft/log/tracker.log`
	• Scanner equivalent `ndtrack.sh` on UNIX-like platforms:
	`/var/tmp/flexera/log/tracker.log`
	 Tip: *If `ndtrack.sh` is executed by a non-root account `userName`, the log defaults to:*
	`/var/tmp/flexera.userName/log/tracker.log`

Example values	`C:\temp\inventory.log`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o LogFile=Inventory.log`

Registry

Installed by	Installation of FlexNet inventory agent
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

LogFile (policy component)

Command line | Registry

`LogFile` gives the location of the log file (on the computer device where the FlexNet inventory agent is executing) when logging is enabled for `mgspolicy` (policy component).

Values

Values / range	Valid log file name, including local and UNC network files.
Default value	`$(TempDirectory)\ManageSoft\policy.log`
Example values	`C:\temp\policy.log`

Command line

Tool	Policy component (`mgspolicy`)
Example	`-o LogFile=policy.log`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)
Computer preference	`[Registry]\ManageSoft\Policy Client\CurrentVersion`

LogFile (upload component)

Command line | Registry

`LogFile` gives the location of the log file (on the computer device where the FlexNet inventory agent is executing) when logging is enabled for `ndupload` (uploader component). All actions performed by `ndupload` are logged to this file. If you wish to use a log file located in a folder other than the default installation log folder, specify a full pathname.

Values

Values / range	Valid log file name, including local and UNC network files.
Default value	`$(TempDirectory)\ManageSoft\uploader.log`
Example values	`C:\temp\MyUploaderLog.log`

Command line

Tool	Uploader component (`ndupload`)
Example	`-o Logfile=C:\temp\myuploaderlogfile.log`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)
Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion`

LogFileOld (application usage component)

Command line | Registry

When the main application usage component log file (defined in LogFile (application usage component)) reaches its maximum size (defined in LogFileSize (application usage component)), the file is renamed to the value in `LogFileOld`. This overwrites any existing `LogFileOld` file. A new log file is created, using the name defined in `LogFile (application usage component)`. This allows you to retain additional log information.

Values

Values / range	Valid file name for local or UNC network files

Default value	Windows devices:
	`$(TempDirectory)\ManageSoft\usageagent.log.old`
	UNIX-like devices:
	`/var/opt/managesoft/log/usageagent.log.old`

Example values	`C:\Temp\usageagent.old` `/tmp/usageagent.old`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o LogFileOld=$(TempDirectory)\usageold.log`

Registry

Installed by	Installation of inventory beacon, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

LogFileOld (installation component)

Command line | Registry

When the main installation component log file (defined in LogFile (installation component)) reaches its maximum size (defined in LogFileSize (installation component)), the file is renamed to the value in `LogFileOld`. This overwrites any existing `LogFileOld` file. A new log file is created, using the name defined in `LogFile (installation component)`. This allows you to retain additional log information.

Values

Values / range	Valid file name for local or UNC network files
Default value	`$(TempDirectory)\ManageSoft\launcher.old.log`
Example values	`$(TempDirectory)\installationold.log`

Command line

Tool	Installation component (`ndlaunch`)
Example	`-o LogFileOld=$(TempDirectory)\installationold.log`

Registry

Installed by	Installation of inventory beacon (computer preference)
Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`

LogFileOld (policy component)

Command line | Registry

When the main policy merging log file (defined in LogFile (policy component)) reaches a particular size (defined in LogFileSize (policy component)), the file is renamed to the value in `LogFileOld`. This overwrites any existing `LogFileOld` file. A new log file is created. This allows you to retain additional log information.

Values

Values / range	Valid file name for local or UNC network files
Default value	`$(TempDirectory)\ManageSoft\policy.old.log`
Example values	`$(TempDirectory)\mgspolicyold.log`

Command line

Tool	Policy component (`mgspolicy`)
Example	`-o LogFileOld=$(TempDirectory)\mgspolicyold.log`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Policy Client\CurrentVersion`

LogFileOld (upload component)

Command line | Registry

When the main upload component log file (defined in LogFile (upload component)) reaches its maximum size (defined inLogFileSize (upload component)), the file is renamed to the value in `LogFileOld`. This overwrites any existing `LogFileOld` file. A new log file is created, using the name defined in `LogFile (upload component)`. This allows

you to retain additional log information.

Values

Values / range	Valid file name for local or UNC network files
Default value	`$(TempDirectory)\ManageSoft\uploader.old.log`
Example values	`$(TempDirectory)\uploaderold.log`

Command line

Tool	Upload component (`ndupload`)
Example	`-o LogFileOld=$(TempDirectory)\uploaderold.log`

Registry

Installed by	Installation of inventory beacon (computer preference)
Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion`

LogFileSize (application usage component)

Command line | Registry

When the main usage component log file (defined in LogFile (application usage component)) reaches the maximum size defined here in `LogFileSize` (application usage component), the file is renamed (to the value in LogFileOld (application usage component)). This overwrites any existing `LogFileOld` (application usage component) file. A new log file is created, with the name defined in LogFile (application usage component). This allows you to retain additional log information.

The size must be expressed as the number of bytes of the maximum allowed log size. If this entry is empty or set to zero, there is no log size limit and the size of the log file continues to grow.

Values

Values / range	Number (number of bytes)
Default value	`524288`

Example values	`3126000` (3 Mb)

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o LogFileSize=1024000`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration.
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion` `[Registry]\ManageSoft\Common`

LogFileSize (installation component)

Command line | Registry

When the main installation component log file (defined in LogFile (installation component)) reaches the maximum size defined here in `LogFileSize (installation component)`, the file is renamed (to the value in LogFileOld (installation component)). This overwrites any existing `LogFileOld` (installation agent) file. A new log file is created, with the name defined in `LogFile (installation component)`. This allows you to retain additional log information.

The size must be expressed as the number of bytes of the maximum allowed log size. If this entry is empty or set to zero, there is no log size limit and the size of the log file continues to grow.

Values

Values / range	Number (number of bytes)
Default value	`4000000`
Example values	`3126000` (3 Mb)

Command line

Tool	Installation component (`ndlaunch`)
Example	`-o LogFileSize=1024000`

Registry

Installed by	Installation of FlexNet inventory agent (adoption) sets the computer preference.
Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`

LogFileSize (policy component)

Command line | Registry

When the main policy merging log file (defined in LogFile (policy component)) reaches the size defined in `LogFileSize`, the file is renamed (to the value in LogFileOld (policy component)). This overwrites any existing `LogFileOld` (policy component) file, A new log file is created. This allows you to retain additional log information.

The size must be expressed as the number of bytes of the maximum allowed log size. If this entry is empty or set to zero, there is no log size limit and the size of the log file continues to grow.

Values

Values / range	Number (number of bytes)
Default value	`4000000`
Example values	`3126000` (3 Mb)

Command line

Tool	Policy component (`mgspolicy`)
Example	`-o LogFileSize=3126000`

Registry

Installed by	Installation of FlexNet inventory agent (adoption)
Computer preference	`[Registry]\ManageSoft\Policy Client\CurrentVersion`

LogFileSize (upload component)

Command line | Registry

When the main upload component log file (defined in LogFile (upload component)) reaches the maximum size defined here in `LogFileSize` (upload component), the file is renamed (to the value in LogFileOld (upload component)). This overwrites any existing `LogFileOld` (upload component) file. A new log file is created, with the name defined in `LogFile` (upload component). This allows you to retain additional log information.

The size must be expressed as the number of bytes of the maximum allowed log size. If this entry is empty or set to zero, there is no log size limit and the size of the log file continues to grow.

Values

Values / range	Number (number of bytes)
Default value	`524288`
Example values	`3126000` (3 Mb)

Command line

Tool	Uploader component (`ndupload`)
Example	`-o LogFileSize=1024000`

Registry

Installed by	Installation of FlexNet inventory agent (adoption) sets the computer preference.
Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion`

LogLevel (inventory component)

Command line | Registry

`LogLevel` sets the category of logging used by the `ndtrack` executable. This logging is output to the file whose name is stored in the `LogFile` preference (see LogFile (inventory component)). Individual entries include the following:

- `A` — Schedule logging
- `C` — Callout logging
- `G` — General logging
- `N` — Network logging
- `P` — Preference logging
- `S` — Security logging
- `U` — User interface logging
- `V` — Verification logging

Values

Values / range	A-z
Default value	`A-z` (this enables all logging)
Example values	`G`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o LogLevel=G`

Registry

Installed by	Installation of the FlexNet inventory agent

Computer preference	In order of precedence: • `[Registry]\ManageSoft\Tracker\CurrentVersion` • `[Registry]\ManageSoft\Common`

LogLevel (policy component)

Command line | Registry

`LogLevel` sets the level of logging used by the policy component. This logging is output to the file whose name is stored in the `LogFile` (policy component) entry (see LogFile (policy component)). Individual entries include the following:

- `A` — Schedule logging
- `C` — Callout logging
- `G` — General logging
- `N` — Network logging
- `P` — Preference logging
- `S` — Security logging
- `U` — User interface logging
- `V` — Verification logging

Values

Values / range	A-z
Default value	`A-z` (this enables all logging)
Example values	`G`

Command line

Tool	Policy component (`mgspolicy`)
Example	`-o LogLevel=G`

Registry

Installed by	Installation of the FlexNet inventory agent
Computer preference	In order of precedence: • `[Registry]\ManageSoft\Policy Client\CurrentVersion` • `[Registry]\ManageSoft\Common`

LogModules (application usage component)

Command line | Registry

`LogModules` specifies the modules used to log events for the application usage component (`mgsusageag` executable). When unspecified, the default inventory agent log modules are used (`default` refers to the default logging within the FlexNet inventory agent). To use custom logging, include the location of your custom logging DLL. Multiple logging DLLs can be specified with a semi-colon separated list.

Values

Values / range	`default; <path to modules>`

Default value	On Microsoft Windows platforms:
	`default;C:\Program Files (x86)\ManageSoft\Common\mgssyslg.dll; C:\Program Files (x86)\ManageSoft\Common\mgsamtlg.dll`
	On UNIX-like platforms:
	`default;`

Example values	`C:\temp\myLogModule.dll`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o LogModules=C:\temp\myLogModule.dll`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

LogModules (inventory component)

Command line | Registry

`LogModules` specifies the modules used to log events for the tracker (`ndtrack` executable). When unspecified, the default inventory agent log modules are used. To use custom logging, include the location of your custom logging DLL.

Values

Values / range	`default; <path to modules>`

Default value	On Microsoft Windows platforms:
	`default;C:\Program Files (x86)\ManageSoft\Common\mgssyslg.dll; C:\Program Files (x86)\ManageSoft\Common\mgsamtlg.dll`
	On UNIX-like platforms:
	`default;$(InstallDir)/lib/levtlog.so`

Example values	`C:\temp\myModule.dll`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o LogModules=C:\temp\myModule.dll`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion` `[Registry]\ManageSoft\Common`

LogModules (policy component)

Command line | Registry

`LogModules` specifies the modules used to log events for the policy component. When unspecified, the default inventory agent log modules are used. To use custom logging, include the location of your custom logging DLL.

Values

Values / range	`default; <path to modules>`

Default value	On Microsoft Windows platforms:
	`default;C:\Program Files (x86)\ManageSoft\Common\mgssyslg.dll; C:\Program Files (x86)\ManageSoft\Common\mgsamtlg.dll`
	On UNIX-like platforms:
	`default;$(InstallDir)/lib/levtlog.so`

Example values	`C:\temp\myModule.dll`

Command line

Tool	Policy component (`mgspolicy`)
Example	`-o LogModules=C:\temp\myModule.dll`

Registry

Installed by	Installation of FlexNet inventory agent
Computer preference	`[Registry]\ManageSoft\Policy Client\CurrentVersion` `[Registry]\ManageSoft\Common`

LowestPriority

Registry

`LowestPriority` specifies the lowest upload/download priority that can be assigned to an inventory beacon. The larger the number, the lower the priority.

When assigning priorities, an inventory device normalizes the calculated priority to fit within the range identified by `HighestPriority` and `LowestPriority`. The lowest priority is commonly set to 100.

Values

Values / range	Recommended 1-100 (but can extend from -2^{31} to 2^{31}).
Default value	No default in registry; default behavior uses 99.
Example values	`80`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\NetSelector\CurrentVersion`

LowProfile (application usage component)

Registry

`LowProfile` determines the CPU priority of the application usage component on the computer device where it is executing as a service. As well as prioritizing the code execution, it simultaneously sets the same priority for all input/output by the usage service on that device.

- When set to `True`, the usage service runs with low priority. For UNIX-like systems, this sets the nice level of the process to 10. On recent Windows platforms, it uses background processing mode (`PROCESS_MODE_BACKGROUND_BEGIN`). On legacy Windows platforms where this is not supported (such as Windows XP and earlier), it uses a priority of idle (`IDLE_PRIORITY_CLASS`).
- When set to `False`, the same processes run with normal priority.

Values

Values / range	Boolean (`True` or `False`).
Default value	No default in registry; default behavior is `True`.
Example values	`False`

Registry

Installed by	Installation of the FlexNet inventory agent
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

LowProfile (inventory component)

Command line | Registry

`LowProfile` determines the CPU priority of the tracker (`ndtrack` executable) on the computer device where it is executing.

- When set to `True`, the tracker processes run with low priority. For UNIX-like systems, this sets the nice level of the process to 10. On recent Windows platforms, it uses background processing mode (`PROCESS_MODE_BACKGROUND_BEGIN`). On legacy Windows platforms where this is not supported (such as Windows XP and earlier), it uses a priority of idle (`IDLE_PRIORITY_CLASS`).
- When set to `False`, the same processes run with normal priority.

Values

Values / range	Boolean (`True` or `False`).
Default value	No default in registry; default behavior is `True`.
Example values	`False`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o LowProfile=True`

Registry

Installed by	Installation of the FlexNet inventory agent
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

MachineID

Command line | Registry

MachineID stores the computer name of the target device, as returned from the operating system (typically shifted to all upper case).

This value is derived from the operating system, and on recent versions of Windows, is the same (possibly apart from case) as the computer name visible in the **Computer > Properties** dialog, displayed as **Computer name**.

By default, this preference is not set manually, but is referenced as $(MachineName). This reference causes the FlexNet inventory agent to query the operating system for the name of the machine.

However, it is possible to override the value with a manual setting. This may be useful, for example, when taking inventory of UNIX-like machines, where you may prefer a particular machine name to appear in inventory. This value may be set:

- In the installation bootstrap file mgsft_rollout_response, used for custom installations on UNIX-like platforms (for more details, see Agent Third-Party Deployment: Configuring the Bootstrap File for UNIX.
- In the command line (on all platforms), as described below.
- In the registry (or, on UNIX-like platforms, for the full FlexNet inventory agent locally installed on the computer, the config.ini file; or when using ndtrack.sh alone as a lightweight inventory scanner, the ndtrack.ini file).

If this preference is customized to a non-default value, it should typically be configured under [Registry]\ManageSoft\Common (only) to ensure that all components use the same value of MachineId. Note that a setting for any individual component overrides the setting in Common.

Values

Values / range	Alphanumeric (best restricted to ASCII characters).
Default value	`$(MachineName)` This variable is expanded by FlexNet inventory agent by querying the operating system.
Example values	`MyMachine`

Command line

Tool	Installation component (ndlaunch), policy component (mgspolicy), scheduling component (ndschedag), usage agent, inventory component (ndtrack)
Example	`-o MachineId=MyMachine`

Registry

Installed by	Installation of FlexNet inventory agent on the target device (by adoption).

Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`
	`[Registry]\ManageSoft\Policy Client\CurrentVersion`
	`[Registry]\ManageSoft\Schedule Agent\CurrentVersion`
	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`
	`[Registry]\ManageSoft\Common`
	(If set in both the `Common` hive and another hive, the value in the `Common` hive is not used for that agent.)

MachineInventoryDirectory

Command line | Registry

`MachineInventoryDirectory` defines the location in which the locally-installed FlexNet inventory agent stores machine inventories.

Note: *The FlexNet inventory agent uses this option only for machine inventory when it is executing on a computer device in local mode. Local mode is set automatically when the base directory for the executable matches the value stored in the registry key `HKLM\Software\ManageSoft Corp\ManageSoft\EtcpInstallDir`. This is the normal state after installation of FlexNet inventory agent on a computer device in the Adopted case. (Conversely, this folder is not used for the Zero-footprint case, for which see MachineZeroTouchDirectory.)*

Values

Values / range	Any valid file path.
Default value	`$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Tracker\Inventories`
Example values	`C:\temp`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o MachineInventoryDirectory=C:\ManageSoft Corp\ManageSoft\Tracker\Inventories`

Registry

Installed by	Installation of the FlexNet inventory agent

Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

MachineName

Command line | Registry

`MachineName` contains the name of the local machine. Unlike `MachineId` , this preference should not be changed.

Values

Values / range	Any valid machine name.
Default value	(No default.) If no value is set, FlexNet Inventory Scanner uses the current machine name.
Example values	`MyMachine`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o MachineName=MyMachine`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

MachinePolicy

Command line | Registry

`MachinePolicy` gives the full directory path and file name of the merged policy file for machine policy on this managed device. This file is prepared on the inventory beacon, and is the summary of all applicable machine policies. The setting can be overridden on the command line for testing purposes, but this setting should generally be read-only.

Values

Values / range	Any valid directory path and file name of a well-formed merged policy file

Default value	`$(MachinePolicyDirectory)\$(MachineId).npl`
Example values	`C:\Document and Settings\All Users\Application Data\Flexera Software\FlexNet Inventory\Policy Client\Policies\Merged\Machine\ WORKSTATION_34.npl`

Command line

Tool	Policy component (`mgspolicy`)
Example	`-o MachinePolicy="C:\temp\testpolicy.npl"`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Selector\CurrentVersion`

MachinePolicyDirectory

Command line | Registry

`MachinePolicyDirectory` gives the location to store the current machine policy on the local managed device.

Values

Values / range	Valid folder and path.
Default value	`$(CommonAppDataFolder)\ManageSoft Corp\ ManageSoft\Policy Client\Policies\Merged\ Machine`
Example values	`C:\Temp\MachinePolicies`

Command line

Tool	Policy component (`mgspolicy`)
Example	`-o C:\Temp\MachinePolicies`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Policy Client\CurrentVersion`

MachinePolicyPackageDirectory

Command line | Registry

`MachinePolicyPackageDirectory` gives the location where package information associated with machine policy is cached.

Values

Values / range	Valid folder and path.
Default value	`$(CommonAppDataFolder)\ManageSoft Corp\ ManageSoft\Policy Client\Packages`
Example values	`C:\Temp\MyMachinePolicy\PackageInfo`

Command line

Tool	Policy component (`mgspolicy`)
Example	`-o C:\Temp\MyMachinePolicy\PackageInfo`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)
Computer preference	`[Registry]\ManageSoft\Policy Client\CurrentVersion`

MachineScheduleDirectory

Command line | Registry

`MachineScheduleDirectory` gives the folder in which the machine schedules are stored.

A machine schedule is run for the computer on which it is installed, regardless of any users that may or may not have accounts on that machine.

Warning: *Altering this value is not recommended. Additional actions need to be taken when redirecting this to another folder. Contact your Flexera professional services consultant for further information.*

Values

Values / range	Valid folder path

Default value	Windows devices:
	`$(CommonAppDataFolder)\ManageSoft Corp\ ManageSoft\Schedule Agent\Schedules`
	Non-Windows devices:
	`$(CommonAppDataFolder)/scheduler/schedules`

Example values	`C:\Program Files\Flexera Software\Schedule Agent\ MachineSchedules`

Command line

Tool	Scheduling component (`ndschedag`)
Example	`-o MachineScheduleDirectory="C:\Program Files\Flexera Software\schedule agent\ MachineSchedules"`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)
Computer preference	`[Registry]\ManageSoft\Schedule Agent\CurrentVersion`

MachineZeroTouchDirectory

Command line | Registry

`MachineZeroTouchDirectory` identifies the Microsoft Windows directory where machine inventory files are written (temporarily, pending upload) in the Zero-footprint case. (This preference is ignored on UNIX-like platforms.)

Note: *The tracker (`ndtrack.exe`) references this setting for any machine inventory involving remote execution (including the Zero-footprint case). Remote mode is set automatically when the registry key `HKLM\Software\ManageSoft Corp\ManageSoft\EtcpInstallDir` does not exist or does not match the base directory for the executable. That registry key is typically missing during Zero-footprint inventory gathering, because the inventory component (tracker) has not been permanently installed on the managed device; so that the Zero-footprint case is classed as remote execution (even though the inventory component has been downloaded and is temporarily running locally on the inventory device). So, in remote mode, the tracker checks for this preference. However, the `MachineZeroTouchDirectory` preference is also typically missing from the registry for the Zero-footprint case, nor is it included in the default command line for this case; with the result that most often, the default value shown below is used. (Conversely, when the full FlexNet inventory agent is locally installed on the managed device, this folder is not used. Instead, for that case see MachineInventoryDirectory.)*

Values

Values / range	Any valid directory.
Default value	`%ProgramData%\ManageSoft Corp\ManageSoft\Tracker\ZeroTouch`
Example values	`C:\temp`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o MachineZeroTouchDirectory=C:\temp`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

ManageSoftPackages

Command line | Registry

`ManageSoftPackages` determines whether or not software inventory is collected for packages installed by the launcher component (`ndlaunch`) on this device. Launcher packages are ignored if this preference is false.

Currently, launcher packages are limited mainly to the upgrade package for the FlexNet inventory agent, and downloaded packages of settings used by the installed FlexNet inventory agent. Therefore this is generally a very low-impact setting; but it is set to false by the command line used for high-frequency hardware checks that may be enabled for subcapacity calculations for IBM PVU licenses.

Values

Values / range	Boolean (`true`/`false`)
Default value	`true`
Example values	`false`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o ManageSoftPackages=false`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

ManualMapperDefaultPriority

Registry

`ManualMapperDefaultPriority` helps to resolve conflicting assignments of an executable (or directory) to an application for which usage is being tracked using the manual mapper.

For application usage tracking, only one application can “own” a file or directory. Where more than one application being tracked specifies the same file or directory, the value of the manual mapper `Priority` preference is used to resolve which application the file or directory is assigned to for tracking. This resolution process occurs at the time the

application is being specified for tracking: the application with the highest value for `Priority` owns the file or directory for usage tracking. Where more than one application specifies the same file or directory for usage tracking, and both have identical priorities, the application whose usage tracking is most recently defined takes precedence.

The value of `ManualMapperDefaultPriority` is assigned to those applications in the manual mapper where no specific `Priority` is assigned.

The default value for this preference, `20`, is set higher than the internal value assigned to other data sources such as Windows Installer, Add/Remove Programs, and so on. These alternate data source are each assigned a priority of 10, meaning that by default, manual mapper entries have priority for specifying usage tracking details. (The order in which usage tracking data is constructed is: the Manual Mapper preference values, native package format, and Add/Remove Programs.)

Values

Values / range	Integer between 1 and 10000.
Default value	`20`
Example value	`100`

Registry

Installed by	Code internals, or manual configuration.
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion\Manual Mapper`

MaximumNetworkRetryPeriod

Command line | Registry

`MaximumNetworkRetryPeriod` specifies the maximum number of seconds to delay between retries when downloading files from a particular inventory beacon. Each time a failure occurs whilst downloading from an inventory beacon, the time to delay before the next retry is increased by NetworkRetryPeriodIncrement (see), up to a maximum value specified by this `MaximumNetworkRetryPeriod`.

Values

Values / range	Numeric (seconds)
Default value	`300`
Example values	`60`

Command line

Tool	Installation component (`ndlaunch`)
Example	`-o MaximumNetworkRetryPeriod=60`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`

MaxKeepAliveLifetime

Command line | Registry

`MaxKeepAliveLifetime` sets the maximum age (in seconds) of a reusable (persistent) HTTP/HTTPS connection. The uploader component (in either FlexNet inventory agent or FlexNet Beacon) does not make further requests on a persistent HTTP/HTTPS upload connection older than this. Instead, it requests a new connection.

Tip: *This setting, which limits the reuse of HTTP/HTTPS connections, is independent of socket-level keep-alive requests that are controlled by SendTCPKeepAlive. Both may operate across the same network connection: for example, when the uploader switches connections to start uploading a distinct file type, this setting may render an older connection unavailable. Otherwise, the settings are independent.*

The special value `MaxKeepAliveLifetime=0` means that there is no maximum age, and the uploader issues keep-alive requests repeatedly (attempting to use the same connection for all file uploads). The same result occurs if you manually remove the registry setting.

The main benefit of limiting the lifetime of a persistent connection is to allow for improved load balancing across large systems. Because load balancers do not re-route persistent connections, dropping the connection and requesting a new one allows for improved load balancing between multiple servers.

Tip: *This `MaxKeepAliveLifetime` setting works in conjunction with `MaxKeepAliveRequests`. If both settings are active, a new HTTP/HTTPS connection is requested (and both counters are reset) for the next upload after either of these limits is reached.*

Values

Values / range	0-2,147,483,647
Default value	`60` seconds (on Windows platforms) `0` (on UNIX-like platforms) – no maximum age by default

Example values	`600`

Command line

Tool	Upload component (`ndupload`)
Example	`-o MaxKeepAliveLifetime=600`

Registry

Installed by	Installer (Windows only), or manual configuration
Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion` `[Registry]\ManageSoft\Common`

MaxKeepAliveRequests

Command line | Registry

`MaxKeepAliveRequests` sets the maximum number of HTTP/HTTPS requests that the uploader component (in either FlexNet inventory agent or FlexNet Beacon) may make on a given persistent HTTP/HTTPS connection. Effectively, this is the number of file uploads attempted before requesting a new connection. The count monitors only uploads (for example, of inventory or discovery files, status reports or logs) and is not affected by downloads (for example, of policy or self-update packages).

Tip: *This setting, which limits the reuse of HTTP/HTTPS connections, is independent of socket-level keep-alive requests that are controlled by SendTCPKeepAlive. Both may operate across the same network connection: for example, when the uploader switches connections to start uploading a distinct file type, this setting may render an older connection unavailable. Otherwise, the settings are independent.*

The special value `MaxKeepAliveRequests=0` means that there is no maximum, and the uploader issues keep-alive requests repeatedly (uses the same connection for all file uploads). The same result occurs if you manually remove the registry setting.

The main benefit of limiting the number of requests on a persistent connection is to allow for improved load balancing across large systems. Because load balancers do not re-route persistent connections, dropping the connection and requesting a new one allows for improved load balancing between multiple servers.

Tip: *This `MaxKeepAliveRequests` setting works in conjunction with `MaxKeepAliveLifetime`. If both settings are active, a new HTTP/HTTPS connection is requested (and both counters are reset) for the next upload after either of these limits is reached.*

Values

Values / range	0-2,147,483,647
Default value	`50` (on Windows platforms) `0` (on UNIX-like platforms)
Example values	`100`

Command line

Tool	Upload component (`ndupload`)
Example	`-o MaxKeepAliveRequests=100`

Registry

Installed by	Installer (Windows only), or manual configuration
Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion` `[Registry]\ManageSoft\Common`

MinInventoryInterval

Command line | Registry

`MinInventoryInterval` specifies the minimum interval (in hours) between collections of inventory. FlexNet inventory agent will not generate or upload inventory if it is invoked less than this period of time after the most recent inventory generation. This preference controls the collection of inventory by the FlexNet inventory agent as well as through zero footprint inventory collection.

The time of the last inventory generation is determined from the last modified time of the most recently cached inventory file, typically stored under `Application Data\ManageSoft Corp\ManageSoft\Tracker\Inventories\` on the inventory device.

Tip: *Since this integer value is specified in hours, it is not possible to specify 30 minutes, which is the maximum period allowed for hardware inventory checking (and uploading where values are changed) when FlexNet Manager Suite is used as a replacement for ILMT for calculating sub-capacity license consumption. In that case, it is necessary to preserve the default value of zero hours minimum.*

Values

Values / range	Any non-negative integer.

Default value	`0`
Example values	`24`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o MinInventoryInterval=24` Generates inventory at most once per day

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

MinRunTime

Command line | Registry

`MinRunTime` specifies the minimum time in seconds that an application must run for before application usage data will be recorded to show that it has been used today. The value must be greater than 0; otherwise the default will be used.

Tip: *This setting determines what the application usage component reports in the uploaded usage data files. After import, the data from all inventory sources (including FlexNet inventory agent) that report application usage data may be filtered by an overriding minimum time set in the* ***Usage*** *tab of each application's properties. For example, suppose the following circumstances:*

- *You set this `MinRunTime=300`, so that on this inventory device, every tracked application that is open for more than 5 minutes is reported as used for today*
- *This usage is visible (after upload to the inventory database) in the* ***Raw Software Usage*** *page of the web interface for FlexNet Manager Suite*
- *Further, let's imagine that the application MyApp was closed after 33 minutes, and that this is the only usage recorded for (say) the last 3 months*
- *Suppose that for MyApp, in the* ***Usage*** *tab of its properties, you set* ***The number of hours an application was active exceeds*** *to 1 hour (in total for the 3 month usage period).*

As a result, while the `MyApp.exe` file may be shown as used in the ***Raw Software Usage*** *page, when you check the* ***Devices*** *tab of the MyApp properties sheet, this inventory device may show a* ***Used*** *value of `No`. In summary, the*

MinRunTime was set sufficiently low to have the day's application run time reported; but the total running time for the usage period was not enough to have MyApp counted as used on this inventory device.

Values

Values / range	Integer greater than 0, being the number of seconds
Default value	`60`
Example values	`300`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o MinRunTime="90"`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

MSI

Command line | Registry

`MSI` applies to inventory for Microsoft Windows devices (and is ignored on UNIX-like platforms):

- When set to `True`, Microsoft Installer (MSI) package information is added to the inventories.
- When set to `False`, the tracker does not include MSI package information in inventories.

Values

Values / range	Boolean (`True` or `False`).
Default value	`True`
Example values	`False`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o MSI=False`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

Name

Registry

`Name` is a human-friendly name for this upload or download record. For example, if the host name is `beacon1.tmnis.org`, you might choose to name this entry `beacon1.tmnis.org Upload Location` or `beacon1.tmnis.org Download Location` respectively.

Values

Values / range	Any name
Default value	(No default.)
Example values	`beacon1.tmnis.org Download Location`

Registry

Installed by	Failover list for inventory beacons, or manual configuration

Computer preference	For uploads:
	`[Registry]\ManageSoft\Common\UploadSettings\<reporting_location>`
	For downloads:
	`[Registry]\ManageSoft\Common\DownloadSettings\<distribution_location>`
	For trusted locations:
	`[Registry]\ManageSoft\Launcher\CurrentVersion\TrustedLocations\<serverkey>`
	For excluded locations:
	`[Registry]\ManageSoft\Launcher\CurrentVersion\ExcludedLocations\<serverkey>`

ndsensNetType

Command line | Registry

`ndsensNetType` applies only on Windows devices. This value determines when a `When connected to network trigger` is deemed to have occurred, causing the command given by `ndsensNetUp` to be executed. It will only trigger if the network is of a certain type. There are three possible values:

- 1 Local area network (LAN)
- 2 Wide area network (WAN)
- 3 Either LAN or WAN.

The scheduling agent monitors the specified network type(s). For example, if `ndsensNetType=2`, the agent only monitors for connections to WANs.

Values

Values / range	1, 2, 3
Default value	3
Example values	1

Command line

Tool	Scheduling component (`ndschedag`)
Example	`-o ndsensNetType=2`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)
Computer preference	`[Registry]\ManageSoft\Schedule Agent\CurrentVersion`

NetworkHighSpeed

Command line | Registry

`NetworkHighSpeed` specifies the lowest measured network speed (in bits per second) that the FlexNet tools will consider to be a high-speed network connection to an inventory beacon. For a full discussion of interacting preferences and tests, see NetworkSpeed.

If `NetworkHighSpeed` has the special (and default) value of `0` (zero), assessment of network bandwidth is terminated, and transfers are attempted at the rate specified by `NetworkMaxRate`.

A non-zero value for `NetworkHighSpeed` acts as the test value for whether a particular network connection is assessed as high speed or low speed. In these cases, transfers are attempted as specified for `NetworkHighUsage` or `NetworkLowUsage`, respectively.

Values

Values / range	Numeric (number of bits per second)
Default value	`0`
Example values	`320`

Command line

Tool	Installation component (`ndlaunch`), inventory component (`ndtrack`), upload component (`ndupload`)
Example	`-o NetworkHighSpeed=320`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	Each tool first checks its own registry entry; and if nothing is found there, next checks `[Registry]\ManageSoft\Common`. The tool-specific registries are: • `[Registry]\ManageSoft\Launcher\CurrentVersion` • `[Registry]\ManageSoft\Tracker\CurrentVersion` • `[Registry]\ManageSoft\Uploader\CurrentVersion`.

NetworkHighUsage

Command line | Registry

`NetworkHighUsage` specifies the maximum percentage of bandwidth that FlexNet tools uses for uploads and downloads on a high-speed connection. The following special cases apply:

- If `NetworkHighUsage` is set to `0`, the installation agent downloads files using 0.1% of the measured bandwidth.
- If `NetworkHighUsage` is outside the range prescribed in `NetworkHighUsageLowerLimit` and `NetworkHighUsageUpperLimit`, it is reset to the nearest range endpoint. For example, consider a case with the following settings:
  - `NetworkHighUsageLowerLimit` = `10`
  - `NetworkHighUsageUpperLimit` = `40`

  If you set `NetworkHighUsage` to 5 (too low), it is automatically reset to 10 (the lower limit). Similarly, if you set `NetworkHighUsage` to 60 (too high), it is automatically reset to 40 (the upper limit).

Use the `Common` key (without any more specialized settings) to keep all tools aligned on the same value.

Values

Values / range	Numeric (percentage 0-100).
Default value	For downloads, `100`. For uploads, no default.
Example values	`55`

Command line

Tool	Installation component (`ndlaunch`), upload component (`ndupload`)

Example	`-o NetworkHighUsage=75`

Registry

Installed by	For downloads, installation of FlexNet inventory agent. For uploads, manual configuration.
Computer preference	For downloads, in order of precedence: • `[Registry]\ManageSoft\Launcher\CurrentVersion` • `[Registry]\ManageSoft\Common`. For data reporting and inventory uploads, in order of precedence: • `[Registry]\ManageSoft\Uploader\CurrentVersion` • `[Registry]\ManageSoft\Common`.

NetworkHighUsageLowerLimit

Command line | Registry

`NetworkHighUsageLowerLimit` specifies the minimum value that can be set for NetworkHighUsage, as described there. The range limits are now useful only for correcting unsuitable values for `NetworkHighUsage`, for example when carelessly specified in a command line. Using the `Common` key (only) causes all tools to adhere to the same standard.

Values

Values / range	Numeric (percentage 0 to 100)
Default value	`100`
Example values	`10`

Command line

Tool	Installation component (`ndlaunch`), inventory component (`ndtrack`), upload component (`ndupload`)
Example	`-o NetworkHighUsageLowerLimit=10`

Registry

Installed by	Installation of FlexNet inventory agent on an inventory device
Computer preference	Each tool first checks its own registry entry; and if nothing is found there, next checks `[Registry]\ManageSoft\Common`. The tool-specific registries are: • `[Registry]\ManageSoft\Launcher\CurrentVersion` • `[Registry]\ManageSoft\Tracker\CurrentVersion` • `[Registry]\ManageSoft\Uploader\CurrentVersion`.

NetworkHighUsageUpperLimit

Command line | Registry

`NetworkHighUsageUpperLimit` specifies the maximum value that can be set for NetworkHighUsage, as described there. The range limits are now useful only for correcting unsuitable values for `NetworkHighUsage`, for example when carelessly specified in a command line. Using the `Common` key (only) causes all tools to adhere to the same standard.

Values

Values / range	Numeric (percentage 0 to 100)
Default value	`100`
Example values	`90`

Command line

Tool	Installation component (`ndlaunch`), inventory component (`ndtrack`), upload component (`ndupload`)
Example	`-o NetworkHighUsageUpperLimit=90`

Registry

Installed by	Installation of FlexNet inventory agent on an inventory device

Computer preference	Each tool first checks its own registry entry; and if nothing is found there, next checks `[Registry]\ManageSoft\Common`. The tool-specific registries are: • `[Registry]\ManageSoft\Launcher\CurrentVersion` • `[Registry]\ManageSoft\Tracker\CurrentVersion` • `[Registry]\ManageSoft\Uploader\CurrentVersion`.

NetworkLowUsage

Command line | Registry

`NetworkLowUsage` specifies the maximum percentage of bandwidth that FlexNet tools use for uploads and downloads on a low-speed connection. The following special cases apply:

- If `NetworkLowUsage` is set to `0`, the installation agent downloads files using 0.1% of the measured bandwidth.
- If `NetworkLowUsage` is outside the range prescribed by `NetworkLowUsageLowerLimit` and `NetworkLowUsageUpperLimit`, it is reset to the nearest range endpoint. For example, consider a case with the following settings:
  - `NetworkLowUsageLowerLimit` = `10`
  - `NetworkLowUsageUpperLimit` = `40`

  If you set `NetworkLowUsage` to 5 (too low), it is automatically reset to 10 (the lower limit). Similarly, if you set `NetworkLowUsage` to 60 (too high), it is automatically reset to 40 (the upper limit).

Tip: *If you wish to use `NetworkLowUsage` to control bandwidth consumption, be sure to modify the default value of NetworkLowUsageLowerLimit.*

Use the `Common` key (without any more specialized settings) to keep all tools aligned on the same value.

Values

Values / range	Numeric (percentage 0-100).
Default value	For downloads, `100`. For uploads, no default.
Example values	`30`

Command line

Tool	Installation component (`ndlaunch`), upload component (`ndupload`)

Example	`-o NetworkLowUsage=50`

Registry

Installed by	For downloads, installation of FlexNet inventory agent. For uploads, manual configuration.
Computer preference	For downloads, in order of precedence: • `[Registry]\ManageSoft\Launcher\CurrentVersion` • `[Registry]\ManageSoft\Common`. For data reporting and inventory uploads, in order of precedence: • `[Registry]\ManageSoft\Uploader\CurrentVersion` • `[Registry]\ManageSoft\Common`.

NetworkLowUsageLowerLimit

Command line | Registry

`NetworkLowUsageLowerLimit` specifies the minimum value that can be set for NetworkLowUsage, as described there. The range limits are now useful only for correcting unsuitable values for `NetworkLowUsage`, for example when carelessly specified in a command line. If you wish to set practical limits on network bandwidth usage by the various tools, you must modify the default value, which otherwise forces all tools to attempt to use all available bandwidth. Using the `Common` key (only) causes all tools to adhere to the same standard.

Values

Values / range	Numeric (percentage 0 to 100)
Default value	`100`
Example values	`10`

Command line

Tool	Installation component (`ndlaunch`), inventory component (`ndtrack`), upload component (`ndupload`)

Example	`-o NetworkLowUsageLowerLimit=10`

Registry

Installed by	Installation of FlexNet inventory agent on an inventory device
Computer preference	Each tool first checks its own registry entry; and if nothing is found there, next checks `[Registry]\ManageSoft\Common`. The tool-specific registries are: • `[Registry]\ManageSoft\Launcher\CurrentVersion` • `[Registry]\ManageSoft\Tracker\CurrentVersion` • `[Registry]\ManageSoft\Uploader\CurrentVersion`.

NetworkLowUsageUpperLimit

Command line | Registry

`NetworkLowUsageUpperLimit` specifies the maximum value that can be set for NetworkLowUsage, as described there. The range limits are now useful only for correcting unsuitable values for `NetworkLowUsage`, for example when carelessly specified in a command line. If you wish to set practical limits on network bandwidth usage by the various tools, you may want to modify the default value to ensure that less than the total available bandwidth is used on low-speed networks. Using the `Common` key (only) causes all tools to adhere to the same standard.

Values

Values / range	Numeric (percentage 0 to 100)
Default value	`100`
Example values	`80`

Command line

Tool	Installation component (`ndlaunch`), inventory component (`ndtrack`), upload component (`ndupload`)
Example	`-o NetworkLowUsageUpperLimit=80`

Registry

Installed by	Installation of FlexNet inventory agent on an inventory device
Computer preference	Each tool first checks its own registry entry; and if nothing is found there, next checks `[Registry]\ManageSoft\Common`. The tool-specific registries are: • `[Registry]\ManageSoft\Launcher\CurrentVersion` • `[Registry]\ManageSoft\Tracker\CurrentVersion` • `[Registry]\ManageSoft\Uploader\CurrentVersion`.

NetworkMaxRate

Command line | Registry

`NetworkMaxRate` sets the number of bytes per second at which the FlexNet tools attempt network transfers (noting that this is specified in bytes and not bits). This preference is not used if:

- `NetworkSpeed` can be determined (for which `NetworkSense` must be `True`)
- `NetworkHighSpeed` has a non-zero value.

In short, `NetworkMaxRate` sets the default bytes per second for network transfers, unless you modify several other default values in preferences.

The special (and default) value of zero means that the tools do not limit transfer rates, and will compete for the maximum available network bandwidth against all other network activity.

Use only the `Common` key to have all tools use the same values with convenient single-point maintenance.

Values

Values / range	Numeric (bytes per second)
Default value	`0` (unlimited)
Example values	`64`

Command line

Tool	Installation component (`ndlaunch`), inventory component (`ndtrack`), upload component (`ndupload`)
Example	`-o NetworkMaxRate=64`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	Each tool first checks its own registry entry; and if nothing is found there, next checks `[Registry]\ManageSoft\Common`. The tool-specific registries are: • `[Registry]\ManageSoft\Launcher\CurrentVersion` • `[Registry]\ManageSoft\Tracker\CurrentVersion` • `[Registry]\ManageSoft\Uploader\CurrentVersion`.

NetworkMinSpeed

Command line | Registry

`NetworkMinSpeed` defines the minimum network speed (in bits per second) for the FlexNet tools to attempt any transfers, including any check for updates to download. For a discussion of the tests and processes, see NetworkSpeed. Using the `Common` key (only) causes all tools to adhere to the same standard.

Values

Values / range	Numeric (bits per second)
Default value	No default in registry; default behavior `1`
Example values	`2000`

Command line

Tool	Installation component (`ndlaunch`), inventory component (`ndtrack`), upload component (`ndupload`)
Example	`-o NetworkMinSpeed=2000`

Registry

Installed by	Code internals, or manual configuration

Computer preference	Each tool first checks its own registry entry; and if nothing is found there, next checks `[Registry]\ManageSoft\Common`. The tool-specific registries are: • `[Registry]\ManageSoft\Launcher\CurrentVersion` • `[Registry]\ManageSoft\Tracker\CurrentVersion` • `[Registry]\ManageSoft\Uploader\CurrentVersion`.

NetworkRetries

Command line | Registry

`NetworkRetries` specifies the number of times a failed network operation is retried before an alternative inventory beacon is contacted. This setting applies to the HTTP and HTTPS protocols (as used by inventory beacons), as well as the FTP protocol (which may be used in your own custom implementations). Note that, for custom implementations, the maximum number of attempts to connect to a file share using the `file:` protocol is controlled by the `ConnectionAttempts` preference (see ConnectionAttempts), and not by this `NetworkRetries` preference.

Values

Values / range	Numeric
Default value	`1`
Example values	`5`

Command line

Tool	Installation component (`ndlaunch`)
Example	`-o NetworkRetries=2`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`

NetworkRetryPeriodIncrement

Command line | Registry

`NetworkRetryPeriodIncrement` specifies the number of seconds to increase each delay between successive attempts to retry a network connection to an inventory beacon. The maximum delay between retries is capped by `MaximumNetworkRetryPeriod` (see MaximumNetworkRetryPeriod). For example, given the following values for these three preferences:

- `NetworkRetries=5`
- `NetworkRetryPeriodIncrement=10`
- `MaximumNetworkRetryPeriod=120`

the delays are increased (from zero) before each retry, giving resultant increments of 10, 20, 30, 40, and 50 seconds. Measuring elapsed time from the initial failure, the retries occur at:

- 10 seconds after initial failure (total delay is 0 + 10)
- 30 seconds after initial failure (total delay is 10 + 20)
- 60 seconds after initial failure (total delay is 30 + 30)
- 100 seconds after initial failure (total delay is 60 + 40)
- 120 seconds after initial failure (total delay is 100 + 50 = 150, but capped at 120 seconds maximum).

The default value of zero for `NetworkRetryPeriodIncrement` means that the retries occur immediately after one another.

Values

Values / range	Numeric (Seconds)
Default value	`0` (zero)
Example values	`10`

Command line

Tool	Installation component (`ndlaunch`)
Example	`-o NetworkRetryPeriodIncrement=10`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration

Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`

NetworkSense

Command line | Registry

`NetworkSense` determines whether network speed checks are performed prior to attempting uploads or downloads.

- When set to true, the various tools that may attempt transfers first check for access and available bandwidth by measuring ping response times for two different sized ping packets. The result is stored in NetworkSpeed, where possible outcomes are discussed. (These speed tests are applied to the top candidate inventory beacon in the fail-over list. These speed tests, and possible network throttling for downloads, do not re-order the fail-over list.)

  Important: *Where ping is disabled or blocked, be sure to use the default (`False`) value for `NetworkSense`. Otherwise the tested connection is deemed to have 0 bps transfer speed, and is discarded from the fail-over list.*

- When set to false (the default), no checks are performed, and the upload or download is attempted immediately.

Because several tools may attempt transfers, this preference can be set in the `Common` key (shown below) so that all tools behave alike. You can also configure an exceptional setting for any one tool (in its `CurrentVersion` key) which overrides the common setting. One scenario you might consider is to have the default `False` value for uploads (since compressed `.ndi` files are small), but set the installation component (`ndlaunch`) preference to `True`, since downloads of agent upgrade packages and the like can be significantly larger.

Values

Values / range	Boolean (`True` or `False`)
Default value	`False`
Example values	`True`

Command line

Tools	Installation component (`ndlaunch`), inventory component (`ndtrack`), upload component (`ndupload`).
Example	`-o NetworkSense=True`

Registry

Installed by	Installation of the FlexNet inventory agent, or manual configuration.
Computer preference	Each tool first checks its own registry entry; and if nothing is found there, next checks `[Registry]\ManageSoft\Common`. The tool-specific registries are: • `[Registry]\ManageSoft\Launcher\CurrentVersion` • `[Registry]\ManageSoft\Tracker\CurrentVersion` • `[Registry]\ManageSoft\Uploader\CurrentVersion`.

NetworkSpeed

`NetworkSpeed` is an internal storage place for the last detected network speed (saved in bits per second).

Note: *Network throttling is not required for normal operations of FlexNet Manager Suite. The throttling controls apply only to the download of packages (such as self-update packages for FlexNet inventory agent in the Adopted case, which are only downloaded when changed, and are rarely updated). For upload of inventories and discovery files, the `.ndi` and `.disco` file formats allow for very efficient compression, giving as much as 90% reduction in file size when compressed for upload. Accordingly, these controls do not provide throttling of uploads. (Speed tests are applied to the top candidate inventory beacon in the fail-over list. These speed tests, and possible network throttling for downloads, do not re-order the fail-over list.)*

The network speed is only assessed when NetworkSense is set to `True`. Before attempting transfers from a new inventory beacon, the applicable tool uses ping packages of different sizes to assess the available bandwidth, saving the result here. This value continues to be used (while `NetworkSense` is true) to manage transfers from this inventory beacon, until a different inventory beacon is chosen, at which time the network speed is reassessed and this value updated. This is then compared with the (non-zero) value of `NetworkHighSpeed` and other preferences as discussed below:

- If `NetworkSpeed` is less than or equal to `NetworkMinSpeed`, no download is attempted. In this case, the connection is 'failed' and the tools assess the next available connection in the fail-over list (see NetworkMinSpeed).
- If `NetworkHighSpeed` has the special (and default) value of `0`, this testing is discontinued, and instead the transfer is attempted at the rate specified in `NetworkMaxRate` (see NetworkMaxRate).
- If `NetworkSpeed` is greater than a non-zero value of `NetworkHighSpeed`, the connection is considered high speed, and the download may use up to the percentage of available bandwidth saved in `NetworkHighUsage` (see NetworkHighUsage).
- If `NetworkSpeed` is non-zero and less than `NetworkHighSpeed`, the transfer may use no more than the percentage of available bandwidth saved in `NetworkLowUsage` (see NetworkLowUsage).
- If ping is disabled or blocked (for example, by a firewall, or because inventory beacons do not respond to ping), the `NetworkSpeed` is deemed to be 0 bits/second, and the connection is discarded from the fail-over list. Testing is ten resumed on the next inventory beacon in the fail-over list.

Important: *Where ping is disabled or blocked, be sure to use the default (`False`) value for NetworkSense.*

Values

Values / range	Numeric (bits per second)
Default value	No default.
Example values	`32000`

Callout reference

Reference as	`$(NetworkSpeed)`

NetworkTimeout

Command line | Registry

`NetworkTimeout` determines the length of time in seconds of inactivity after which a network operation will time out. Notice that the sending of keep-alive TCP packets is ignored for this purpose — these do not count as network activity.

Values

Values / range	String (integer value of seconds)
Default value	`600` (ten minutes)
Example values	`1200`

Command line

Tool	Upload component (`ndupload`)
Example	`-o NetworkTimeout=1000`

Registry

Installed by	Installer, or manual configuration

Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion` `[Registry]\ManageSoft\Common`

no_proxy

Command line | Registry

`no_proxy` lists the addresses for which the installation agent should ignore the proxy settings entered in the `http_proxy` registry entry.

Values

Values / range	Any valid URL.
Default value	(No default.)
Example values	`tmnis.com;tmnis.com.de`

Command line

Tool	Installation component (`ndlaunch`)
Example	`-o no_proxy=tmnis.com;tmnis.com.de`

Registry

Installed by	Installation of FlexNet inventory agent on a managed device (computer preference)
Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`

OnConnect

Command line only

`OnConnect` applies only on Windows devices. Determines scheduling tasks that are initiated when an `OnConnect` trigger occurs. There are two options:

- `True`: Run missed tasks (automatically setting the preference `Catchup=Always`) and run all events with `OnConnect` triggers
- `False`: Do nothing.

Values

Values / range	Boolean (`True` or `False`)
Default value	`False`
Example values	`True`

Command line

Tool	Scheduling component (`ndschedag`)
Example	`-o OnConnect=True`

OnlyGenerateIfHardwareChanged

Command line | Registry

`OnlyGenerateIfHardwareChanged` is a preference used to manage the upload of hardware inventory, particularly for use in high-frequency subcapacity license calculations for IBM PVU licenses. IBM may grant a license variation allowing the use of subcapacity calculations by FlexNet Manager Suite, for which typical conditions include a requirement for 30 minute checks on hardware changes and virtual machine relocations. To minimize both the performance impact on the target inventory devices and network load from inventory uploads, this setting allows for the upload of the inventory (`.ndi`) file only when the hardware inventory has changed since the last inventory file upload. The effects of the setting values are:

- False means that a hardware inventory (`.ndi`) file is uploaded from this inventory device to an appropriate inventory beacon after every inventory check (typically, every 30 minutes).
- True means that, when nothing has changed since the last upload of an inventory file, no new upload occurs; but if there is any change in the relevant hardware characteristics (including the location of virtual machines, where relevant), a new `.ndi` file for the entire current state of the hardware is uploaded. You may also specify some WMI properties where changes are irrelevant and ignored (see HardwareChangesClassPropertyBlacklist).

Values

Values / range	Boolean (`true`/`false`)
Default value	`false`
Example values	`true`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o OnlyGenerateIfHardwareChanged=true`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

OracleEnvironmentCmdTimeoutSeconds

Command line | Registry

`OracleEnvironmentCmdTimeoutSeconds` applies when the tracker issues commands to gather inventory on Oracle Database and associated options. Some of these commands may hang – for example, with some commands, some machines are configured to prompt for a password. Since `ndtrack` is unable to supply credentials, it would previously hang, and the overall inventory gathering process would therefore fail.

With the addition of a timeout, `ndtrack` can now time out on the blocked command, so that only the Oracle inventory gathering step fails, and the rest of the process can continue.

Tip: *This option is available only on UNIX-like platforms, and if you wish to override the default value, it may be set either on the command line or in the `config.ini` file that functions as a pseudo-registry on those platforms.*

Values

Values / range	`1` through `60` (inclusive, in seconds). Numeric values outside this range are corrected to the nearest boundary value.
Default value	`10` (This is the default value when there is no entry, or a non-numeric entry in the command line options.)
Example value	`20`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o OracleEnvironmentCmdTimeoutSeconds=20`

Registry

Installed by	Manual configuration (computer preference)
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

OracleInventoryAsSysdba

Command line | Registry

`OracleInventoryAsSysdba` applies on UNIX-like platforms only. It modifies the command line used for the Oracle utility `sqlplus` during inventory gathering, allowing use of a custom account name.

Tip: *This preference only controls the command line to be used in the event that gathering Oracle inventory is permitted by other preferences:*

- *PerformLocalScripting*
- *PerformOracleInventory*
- *IncludeLocalScriptRule*
- *ExcludeLocalScriptRule*.

The effect of `OracleInventoryAsSysdba` is:

- When `OracleInventoryAsSysdba=True` (or when the value is not specified), the inventory component (`ndtrack`) uses the following command to access the Oracle Database for inventory gathering:

  ```
  sqlplus "/ as sysdba"
  ```

- When `OracleInventoryAsSysdba=False`, the inventory component (`ndtrack`) uses the following:

  ```
  sqlplus "/ "
  ```

  Notice the trailing white space in this case.

Tip: *When `OracleInventoryAsSysdba=False`, it is mandatory to set an account name, using `OracleInventoryUser` (see OracleInventoryUser).*

Values

Values / range	Boolean (`True` or `False`)
Default value	`True` (This is the default value when there is no entry in the registry file or command line options.)
Example values	`False`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o OracleInventoryAsSysdba=False`

Registry

Installed by	Manual configuration (computer preference)

Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion` in the `config.ini` or `ndtrack.ini` file:
	`[ManageSoft\Tracker\CurrentVersion]`
	`OracleInventoryAsSysdba=False`

OracleInventoryUser

Command line | Registry

`OracleInventoryUser` specifies an operating system account which the FlexNet inventory agent may use when collecting inventory data from an Oracle Database on a UNIX host. This preference applies only to UNIX-like platforms.

(On Microsoft Windows, `ndtrack` executes as `SYSTEM`, which must be a member of the `ora-dba` database group on the Windows Server. For manual execution on the command line in Microsoft Windows, you can alternatively specify another account that has administrator privileges and is a member of `ora-dba`.)

On UNIX-like platforms, there are additional requirements. FlexNet inventory agent must first use the operating system account named in `OracleInventoryUser` to invoke the Oracle-supplied `sqlplus` executable, and then (without requiring additional passwords) have `sqlplus` connect to the database instance to gather inventory. In order for the database instance to trust the `OracleInventoryUser` account without additional passwords, the database instance must be configured to allow operating system (OS) authentication for that account.

- Giving the `OracleInventoryUser` account the necessary privileges to launch `sqlplus` is typically achieved by including the account name in the operating system group owning Oracle inventory (by default called `oinstall`, named at Oracle installation) for each installation of Oracle Database.
- Configuring for OS authentication requires that you check the OS authentication prefix (default `ops$`) for this database instance, and then register the account with a command like:

  ```
  CREATE USER ops$OracleInventoryUser IDENTIFIED EXTERNALLY;
  ```

  of course substituting the actual account name for the placeholder (for more details, see your Oracle documentation).
- Having `sqlplus` (running as `OracleInventoryUser`) successfully connect to the database instance can be achieved in either of these two ways:

- When `OracleInventoryAsSysdba=True` (or is unspecified, since the default is `True`), the `OracleInventoryUser` account must also be a member of the Oracle `dba` group. In this case, `ndtrack` logs into the database instance with the command:

  ```
  sqlplus "/ as sysdba"
  ```

  Note: *When `OracleInventoryAsSysdba=True` but `OracleInventoryUser` is not set, `ndtrack` analyzes the services process listings to extract the SID of running instances of Oracle Database. It then identifies the user account that launched that process (which account is assumed to be a member of the `dba` group), and uses the same account name to take inventory of the database instance.*

- When `OracleInventoryAsSysdba=False`, the OS account need not be in the `dba` group, but must still be registered for OS authentication (as noted above). As an Oracle user, this account must also have appropriate permissions for inventory gathering (for table-level permissions, see *Appendix C: Oracle Tables and Views for Oracle Inventory Collection* in *FlexNet Manager Suite System Reference*). In this case, `ndtrack` logs into the database instance with the command:

  ```
  sqlplus "/ "
  ```

Tip: *If you are setting `OracleInventoryUser`, and the path used to start the target Oracle database instance included a symbolic link, you might also consider whether to set an environment variable for this account that identifies the `ORACLE_HOME` for the target database instance. For more insight, see UserDefinedOracleHome.*

Warning: *The user name of the operating system account must not include a hash (#) character, as this causes a failure when attempting to upload the generated `.ndi` files to the application server.*

Values

Values / range	An exact match for any Oracle user name that: • Is also an operating system account • Has OS authentication enabled • Is a member of `oinstall` (or equivalent group, granting execute permissions for `sqlplus`) • Is either a current member of the `dba` group on the UNIX host server; or has adequate permissions for inventory gathering (check descriptions above).
Default value	None. (See note above for impacts when this preference is not specified.)
Example value	`dbauser`

Command line

Tool	Either: • FlexNet inventory agent (`ndtrack`) locally installed on the UNIX server (the preference may be on the command line or in `config.ini`) • The tracker (`ndtrack.sh`) executed remotely from an inventory beacon in the zero footprint inventory collection case (the preference may be on the command line or in `ndtrack.ini`).
Example	`-o OracleInventoryUser=dbauser`

Registry

Installed by	Manual configuration

Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion` in the `config.ini` or `ndtrack.ini` file:
	`[ManageSoft\Tracker\CurrentVersion]`
	`OracleInventoryUser=dbauser`

Password

Registry

`Password` stores the encrypted password required for authentication during upload of files from the managed device to the inventory beacon.

Values

Values / range	This can be any characters. The characters are encrypted.
Default value	For FT protocol only: `Anonymous` Otherwise, there is no default.

Registry

Installed by	Failover list for inventory beacons, or manual configuration

Computer preference	For uploads:
	`[Registry]\ManageSoft\Common\UploadSettings\<reporting_location>`
	For downloads:
	`[Registry]\ManageSoft\Common\DownloadSettings\<distribution_location>`

PerformDockerInventoryScan

Command line | Registry

`PerformDockerInventoryScan` enables inventory to be collected for Docker containers on Linux and Windows Server 2016 64 bit hosts. This includes information for the Docker container host, container images, and associated containers. Application inventory is collected from running containers. The Docker service must be available on the local host through the default Docker socket. The agent component will not continue to run if it cannot connect the Docker service.

The Docker agent component gathers inventory for an installed container image by injecting the Zero-footprint inventory tracker into a running container based on that image. Once an inventory for that image has been successfully collected the agent will not inject the tracker into any subsequent containers based on that image. The image is identified by its ID, rather than its tag, so updating a particular image will result in the agent injecting the tracker into containers launched from the updated image.

Note: *The agent requires a running container instance to gather inventory, and so will not gather inventory for installed container images for which a container has not been launched.*

Values

Values / range	Boolean (`True` or `False`)
Default value	`False`
Example value	`True`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o PerformDockerInventoryScan=True`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration.

Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion` in the `config.ini` or `ndtrack.ini` file:
	`[ManageSoft\Tracker\CurrentVersion]`
	`PerformDockerInventoryScan=True`

PerformIBMDB2Inventory

Command line | Registry

`PerformIBMDB2Inventory` controls whether or not the locally-installed FlexNet inventory agent utilizes the `db2licm` command to improve inventory recognition of IBM Db2 database and its available add-ons. This command is a standard part of the IBM Db2 installation, and requires no special configuration or command-line options for its execution.

Tip: *On UNIX-like platforms, the FlexNet inventory agent must run with elevated privileges in order to execute the `db2licm` command. The default configuration, where the FlexNet inventory agent runs as `root`, allows for execution of this command. If this `PerformIBMDB2Inventory` is set to `True` but the FlexNet inventory agent is not running with elevated privileges, there are various results:*

- *The feature to collect additional Db2 inventory is disabled internally, and [Registry] or command-line settings are ignored*
- *The following is added to the tracker log file (and further detail is available in `mgstrace.log`, if you are running the trace file):*

  ```
  IBM Db2 inventory is disabled as the inventory agent is not running in the administrator's context.
  ```

These conditions do not apply to Microsoft Windows platforms, where IBM Db2 allows a standard user account to run the `db2licm.exe` command without error.

Running `db2licm` and associated commands allows the imported inventory to populate the **IBM Db2 Database and Add-Ons** report with product details and license consumption results. If these command cannot be used (for example, because the `PerformIBMDB2Inventory` preference has been set to `False`), the report cannot be populated.

Values

Values / range	Boolean (`True` or `False`)

Default value	`True` This default is automatically applied when there is no value available in the [Registry] or command line.
Example values	`False`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o PerformIBMDB2Inventory=False`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

PerformIBMWebSphereMQScan

Command line | Registry

`PerformIBMWebSphereMQScan` determines whether the inventory component tests the local inventory device for an active message queue managed by IBM MQ (previously known as IBM WebSphere MQ). A license entitlement for IBM MQ is consumed only where there is an active message queue.

This preference is set to false in the tracker command line used for the high-frequency hardware checking required for subcapacity calculations for IBM PVU licenses.

Values

Values / range	Boolean (`true`/`false`)
Default value	`true` This is the value used for machine-based inventory when the preference has not been set or declared in a command line.
Example values	`false`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o PerformIBMWebSphereMQScan=false`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

PerformKvmInventory

Command line | Registry

`PerformKvmInventory` specifies whether or not the inventory component, reporting on a Linux host server that is using Kernel-based Virtual Machine (KVM) virtualization, should also report the presence of guest VMs as part of its uploaded inventory. This preference works with either a locally-installed inventory component (such as the full FlexNet inventory agent), or with Zero-footprint remote inventory gathered by an inventory beacon.

When inventory has been uploaded from the host and from all virtual machines, the additional list of guest OS devices on the host allows FlexNet Manager Suite to correctly link the host and VMs for license calculations (some license types, such as IBM PVU licenses, take the capacity of the host into account when licensing software running on the guest VMs). Since the mapping of guests to host is important for correct license consumption, this preference defaults to `True` even when the preference is not specified in the `config.ini` file (the pseudo-registry for non-Windows platforms).

The preference has no effect on other systems (for example, it does not affect Windows devices, and does nothing on VMs or stand-alone Linux devices that are not hosting KVM guests).

Values

Values / range	Boolean (`True` or `False`)
Default value	No default in registry; default behavior is `True`
Example values	`False`

Command line

Tool	Inventory component (`ndtrack`)

Example	`-o PerformKvmInventory="False"` On a KVM host server, setting this command-line option for the inventory component to `False` prevents the host identifying its guest virtual machines within inventory. Normal inventory of hardware, and software locally installed on the host server itself, is still reported. When the guest VMs are not identified because of this `False` setting, the host server cannot be displayed in inventory as a `VMHost`, and instead its **Inventory device type** is shown as `Computer`. (Conversely, omitting the option has the same effect as setting it to `True` – that is, the host reports its guest VMs along with its other inventory, and the host server can now be correctly identified as a `VMHost`.)

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

PerformLocalScripting

Command line | Registry

Tip: *This preference requires that the up-to-date `InventorySettings.xml` file is either:*

- *Co-located with the FlexNet Inventory Scanner (or the scanner-like `ndtrack.sh` on UNIX-like platforms)*
- *Correctly installed with the fully-installed FlexNet inventory agent.*

Unless this condition is met, this preference is ignored.

By default when the above condition is met, the tracker (`ndtrack` executable) will perform all local scripting actions specified in `InventorySettings.xml`. For example, this file includes enhanced inventory abilities related to Oracle databases and the hardware they run on, and Microsoft Exchange (although the functionality available is subject to the products you have licensed within FlexNet Manager Suite). When false, the `PerformLocalScripting` option prevents execution of any of the scripting enabled in the `InventorySettings.xml` file.

Tip: *Using either the `IncludeLocalScriptRule` or `ExcludeLocalScriptRule` preference, you can separately manage local scripting by leaving some enhanced inventory collection operational.*

Note: *Collecting inventory from local Oracle Database instances requires that **none** of the following conditions apply:*

- *`PerformLocalScripting=False` (this turns off all scripts, including Oracle inventory)*
- *`PerformOracleInventory=False` (this turns off Oracle inventory)*
- *`ExcludeLocalScriptRule="OracleRule"`*

- *`IncludeLocalScriptRule` is set to any value that does not include `OracleRule` (which is thereby excluded).*

When all these preferences have their default values, inventory gathering from local Oracle Database instances can proceed.

Values

Values / range	Boolean (`True` or `False`)
Default value	`True`
Example values	`False`

Command line

Tool	`ndtrack`
Example	`-o PerformLocalScripting=False`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\Managesoft\Tracker\CurrentVersion`

PerformOracleFMWScan

Command line | Registry

***Tip:** This preference requires that:*

- *The FlexNet Manager for Datacenters product has been licensed*
- *The up-to-date `InventorySettings.xml` file (version 56 or later) is either:*
  - *Co-located with the FlexNet Inventory Scanner (or the scanner-like `ndtrack.sh` on UNIX-like platforms)*
  - *Correctly installed with the fully-installed FlexNet inventory agent.*

Unless both of these conditions are met, this preference is ignored.

Oracle Fusion Middleware is a large collection of software products used for the development, deployment, and management of software services. Oracle have specific conditions for tracking licensing for Fusion Middleware, and FlexNet Manager Suite is verified by Oracle as an acceptable tool to collect the appropriate inventory.

By default, the tracker (`ndtrack` executable) assumes the value `False` for this preference, and therefore takes no related action. However, if an operator sets the **Enable collection of Oracle Fusion Middleware audit data** check box (located in **Discovery & Inventory > Inventory Settings > Oracle audit data scanning**), the change is distributed through device policy to the locally-installed FlexNet inventory agent on all managed inventory devices. When the check box selection arrives through device policy, it is saved by setting this preference to `True` in the registry, as shown in the table below. Alternatively (as is typical for preferences), the preference may be set in the registry manually or through the use of a third-party registry management tool; or it may be set in the command line for the tracker.

Important: *When `PerformOracleFMWScan` is set, the following preferences also take effect:*

- *PerformLocalScripting must be true (this is its default value).*
- *Optionally, ExcludeDirectory may be used, for example, to filter out directories that are not required for scanning for Oracle Fusion Middleware.*

Once `PerformOracleFMWScan` is set to `True` (and its prerequisites are correctly set), the FlexNet inventory agent (version 17.0.1 or later) collects evidence, along with certain additional files that Oracle requires for Oracle Fusion Middleware. This content is organized into a specific structure, and then archived, and included as a blob of binary data within the standard archived `.ndi` file for each inventory upload, and eventually imported into FlexNet Manager Suite through the standard processes.

Remember: *By default, the uploaded data is only available within FlexNet Manager Suite, and not included in the nightly archive saved for possible submission to Oracle. A separate check box,* ***Include Oracle Fusion Middleware*** *(on the* ***Inventory*** *tab of the* ***System Settings*** *page) allows the uploaded data and files to be incorporated into the `OracleGLASEvidence.zip` archive, ready for submission when an audit is required. As well, the uploaded data for Oracle Fusion Middleware is converted into installer evidence, for presentation in the web interface of FlexNet Manager Suite, and inclusion in the nightly license compliance calculations.*

Tip: *As well as this preference and its prerequisites mentioned above, any of the following preferences may also affect collection of regular Oracle inventory. However, none of the following have any impact on the collection of Oracle Fusion Middleware inventory, controlled through `PerformOracleFMWScan`:*

- *PerformOracleInventory*
- *PerformOracleListenerScan*
- *ExcludeLocalScriptRule — has no effect provided that it does* ***not*** *list the rules controlling Oracle Fusion Middleware inventory collection (`OracleFMWBase`, `OracleFMWRule`, or `OracleCPURule`)*
- *IncludeLocalScriptRule — has no effect either when it is not set, or is set to a value that* ***includes*** *the three rule names mentioned above (recall that this is an exclusive list when it is set, so that anything not included in its value is effectively excluded).*

Values

Values / range	Boolean (`True` or `False`)

Default value	`False`

Tip: *If the preference has not been set in the registry following the download of device policy, this default value is supplied by the tracker internally.*

Example values	`True`

Command line

Tool	`ndtrack`
Example	`-o PerformOracleFMWScan=true`

Registry

Installed by	Either: • Manual configuration (without which, code internals supply the default) • Download of device policy that sets the value to `True`.
Computer preference	`[Registry]\Managesoft\Tracker\CurrentVersion`

PerformOracleInventory

Command line | Registry

Tip: *This preference requires that:*

- *The FlexNet Manager for Datacenters product has been licensed*
- *The up-to-date `InventorySettings.xml` file is either:*
  - *Co-located with the FlexNet Inventory Scanner (or the scanner-like `ndtrack.sh` on UNIX-like platforms)*
  - *Correctly installed with the fully-installed FlexNet inventory agent.*

Unless both of these conditions are met, this preference is ignored.

By default when the above conditions are met, the tracker (`ndtrack` executable) tests for, and if possible collects, Oracle Database instance inventory each time an inventory collection job is scheduled. By setting the value to `false`, you can use the `PerformOracleInventory` option to prevent the tracker from collecting Oracle Database inventory on an individual target device. This preference affects only the collection of inventory from Oracle Database instances, and does not control the collection of Oracle hardware information.

Tip: *Collection of Oracle inventory may be affected by any of the following preferences:*

- *PerformLocalScripting*
- *PerformOracleInventory*
- *PerformOracleListenerScan*
- *ExcludeLocalScriptRule*
- *IncludeLocalScriptRule*.

Values

Values / range	Boolean (`True` or `False`)
Default value	`True`
Example values	`False`

Command line

Tool	`ndtrack`
Example	`-o PerformOracleInventory=False`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\Managesoft\Tracker\CurrentVersion`

PerformOracleJavaAuditScan

Command line | Registry

Tip: *This preference requires that the up-to-date `InventorySettings.xml` file is either:*

- *Co-located with the FlexNet Inventory Scanner (or the scanner-like `ndtrack.sh` on UNIX-like platforms)*
- *Correctly installed with the fully-installed FlexNet inventory agent.*

Unless this condition is met, this preference is ignored.

The Oracle Java audit scan collects additional information on the Java installations found on the system being scanned - such as build type (for example, `commercial`), installation date, presence of the release file, and publisher. These

additional properties together with some hardware information from the system are incorporated into the `OracleGLASEvidence.zip` archive, ready for submission when an audit is required.

By default, the tracker (`ndtrack` executable) assumes the value `False` for this preference, and therefore takes no related action. However, if an operator sets the **Enable collection of Oracle Java audit data** check box (located in **Discovery & Inventory > Inventory Settings > Oracle audit data scanning**), the change is distributed through device policy to the locally-installed FlexNet inventory agent on all managed inventory devices. When the check box selection arrives through device policy, it is saved by setting this preference to `True` in the registry, as shown in the table below. Alternatively (as is typical for preferences), the preference may be set in the registry manually or through the use of a third-party registry management tool; or it may be set in the command line for the tracker.

Important: *For the `PerformOracleJavaAuditScan` preference to take effect, the following preferences must also be set:*

- *PerformLocalScripting must be true (this is its default value).*
- *IncludeDirectory must not be turned off through the web interface of FlexNet Manager Suite. This is controlled through the **Inventory Settings** page (navigate to **Discovery & Inventory** > **Settings**, and expand **File evidence**), where for each type of computer platform, there are three options:*
  - ***Do not collect file evidence** — **Do not** select this option, since turning off collection of file evidence prevents gathering of inventory for Oracle Java audit.*
  - ***Collect file evidence for all folders** — This setting requires a careful balancing act:*
    - *On the one hand, use this option with caution, since its effect is not limited to Java products. It may cause very large increases in total file evidence collected, uploaded, and processed during license reconciliation, and may therefore have major impacts on system performance. (However, the good news is that, if you use this setting temporarily and then turn if off again, excess uploaded file evidence that subsequently disappears from future inventory uploads is automatically cleaned up in the next compliance calculation.)*
    - *On the other hand, Oracle requires that all folders are scanned, to ensure that no installations of Java are overlooked.*
  - ***Collect file evidence for specified folders** — You may use this option to specify as many directories as needed to ensure coverage of all your Java installations. Keep in mind that these settings are distributed through policy to all managed inventory devices (those getting policy from an inventory beacon and uploading to one as well) throughout your computing estate. The option is therefore best suited if you distribute a Standard Operating Environment (SOE) to your servers, so that Java products are installed in folders consistently named per platform across your enterprise.*
- *Optionally, as always, ExcludeDirectory may be used, for example, to filter out child directories that are not required from the parents included in the previous settings.*

Once `PerformOracleJavaAuditScan` is set to `True` (and its prerequisites are correctly set), the FlexNet inventory agent (version 18.4.0 or later) collects the inventory that Oracle requires for Java audits. The collected inventory is included with the standard archived `.ndi` file for each inventory upload, and eventually imported into FlexNet Manager Suite through the standard processes.

Remember: *By default, the uploaded data is only available within FlexNet Manager Suite, and not included in the nightly archive saved for possible submission to Oracle. A separate check box, **Include Oracle Java** (on the **Inventory** tab of the **System Settings** page) allows the uploaded data and files to be incorporated into the*

OracleGLASEvidence.zip archive, ready for submission when an audit is required.

Important: *On the Windows platform, only the java.exe executables that are digitally signed will be processed. For the java.exe executables that are not digitally signed, file evidence will still be reported, but the executables will not be executed to obtain version information, nor will additional Java verification checks be performed. For details about command line parameters, see Common: Child Processes on Windows Platforms.*

Values

Values / range	Boolean (True or False)
Default value	False ***Tip:*** *If the preference has not been set in the registry following the download of device policy, this default value is supplied by the tracker internally.*
Example values	True

Command line

Tool	ndtrack
Example	-o PerformOracleJavaAuditScan=true

Registry

Installed by	Either: • Manual configuration (without which, code internals supply the default) • Download of device policy that sets the value to True.
Computer preference	[Registry]\Managesoft\Tracker\CurrentVersion

PerformOracleListenerScan

Command line | Registry

When PerformOracleListenerScan is True (or not set), the inventory component (ndtrack) checks process listings to discover Oracle listeners running on the local device, and if any are found, prepares a .disco file listing all Oracle Database instances identified by the local Oracle listener(s) (the database instances themselves may be on the

same local device, or on a different server). When this preference is false, or when no listeners are found, no Oracle `.disco` file is prepared for the local device.

Tip: *This `.disco` file is not a prerequisite for gathering local Oracle Database inventory. The inventory component (`ndtrack`) takes inventory of local Oracle Database instances based on the `PerformOracleInventory` preference. These two controls (and processes) are independent.*

Values

Values / range	Boolean (`True` or `False`)
Default value	`True`
Example values	`PerformOracleListenerScan=False`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o PerformOracleListenerScan=False`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

PerformSymantecSFScan

Command line | Registry

`PerformSymantecSFScan` controls whether or not the inventory component (the tracker) gathers inventory for Veritas Storage Foundation products installed on the local inventory device. (Since January 2016, Veritas has operated independently from Symantec, but the preference name is maintained for backwards compatibility.)

Generally a low impact preference, `PerformSymantecSFScan` is set to false by the tracker command line used for high-frequency hardware inventory checks required for subcapacity calculations for IBM PVU licenses.

Values

Values / range	Boolean (`true`/`false`)

Default value	`true` This is the value used for machine-based inventory when the preference has not been declared.
Example values	`false`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o PerformSymantecSFScan=false`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

PerformVirtualBoxInventory

Command line | Registry

`PerformVirtualBoxInventory` determines whether the FlexNet inventory agent attempts to detect and scan for a `VirtualBox` instance on the machine and then identify **Oracle VM VirtualBox Extension Pack**. It does this by scanning the following locations (in order):

1. (Windows) `%ProgramFiles%\Oracle\VirtualBox`
2. (Linux) `/opt/VirtualBox`
3. (Linux) `/usr/lib/virtualbox`
4. (Solaris) `/opt/VirtualBox/amd64`
5. (Solaris) `/opt/VirtualBox/i386`
6. (MacOS) `/Applications/VirtualBox.app/Contents/MacOS`
7. (FreeBSD) `/usr/local/lib/virtualbox`
8. (OS/2) `C:/Apps/VirtualBox`
9. (All) `%VBOX_APP_HOME%\`
10. (Windows) `%VBOX_MSI_INSTALL_PATH%`

11. (Windows) %VBOX_INSTALL_PATH%

Tip: *As usual, values shown between percent signs (such as %ProgramFiles% or %VBOX_APP_HOME%\) are system environment variables that can be set on the device where the FlexNet inventory agent is running.*

The FlexNet inventory agent is searching for the VBoxManage executable, which is Oracle's own command-line interface for Oracle VM VirtualBox used to identify extension packs. When it finds VBManage.exe it runs the command VBManage.exe list extpacks to retrieve this list.

Values

Values / range	Boolean (True or False)
Default value	No default in registry; default behavior is True
Example values	False

Command line

Tool	Inventory component (ndtrack)
Example	-o PerformVirtualBoxInventory="False"

Registry

Installed by	Code internals, or manual configuration
Computer preference	[Registry]\ManageSoft\Tracker\CurrentVersion

PlatformSpecificPackages

Command line | Registry

PlatformSpecificPackages specifies whether software inventory gathering should seek information about non-Windows, platform-specific packages (for example lpp, pkg, rpm and sd-ux). This setting is ignored on Windows devices.

Values

Values / range	Boolean (true/false)

Default value	`true` This value is used when no preference is specified in `config.ini`.
Example values	`false`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o PlatformSpecificPackages=false`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

PolicyPackageRefreshPeriod

Registry

`PolicyPackageRefreshPeriod` specifies the number of seconds after successfully downloading package (`.osd`) files during which download of those files should not be attempted again.

If a value is set for this preference, each time policy is applied the policy agent checks to see if package files required by policy have been downloaded within this time period. If so, the currently downloaded package files are used for installation.

If package files have not been downloaded within the time interval, they are remotely checked for changes since they were last downloaded to the inventory device, and only downloaded if changed. (The method of checking this depends on the protocol in use. For HTTP downloads, an `If-Modified-Since` HTTP request is used. Equivalent requests are made for other protocols.)

If *no* value is set for this preference, package files are always downloaded, regardless of whether or not they have changed since they were last downloaded.

If, instead, you always want *newer* package files (those that have changed since the last download to the inventory device) to be downloaded when policy is applied, set the value of this preference to `0` (zero).

Values

Values / range	Integer in the range 0 - 1000000000

Default value	`86400` 86400 seconds is 24 hours
Example values	`28800` 28800 seconds is 8 hours

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Common`

PolicyServerURL

Command line | Registry

`PolicyServerURL` gives the location of policy for server-side merging. The policy agent passes this path through to the installation agent.

Warning: *Internal use only: do not edit.*

Values

Values / range	Valid URL.
Default value	`http://beacon.domain.com/ManageSoftDL/Policies/Merged/_domain/Machine/deviceHostname.npl`
Example values	`http://InvBeaconB/ManageSoftDL/Policies/Merged/Machine/Domain/MgdDev2.npl`

Note: *`_domain` is replaced with the actual domain of the device when known.*

Command line

Tool	Policy component (`mgspolicy`)

Example	`-o PolicyServerURL=http://InvBeaconB/ManageSoftDL/Policies/Merged/Machine/Domain/MgdDev2.npl`

Registry

Installed by	Code internals
Computer preference	`[Registry]\ManageSoft\Policy Client\CurrentVersion`

Port

Registry

`Port` is the port number for data transfer.

Values

Values / range	Integers
Default value	(No default.)
Example values	1099

Registry

Installed by	Failover list for inventory beacons, or manual configuration

Computer preference	For uploads:
	`[Registry]\ManageSoft\Common\UploadSettings\<reporting_location>`
	For downloads:
	`[Registry]\ManageSoft\Common\DownloadSettings\<distribution_location>`
	For trusted locations:
	`[Registry]\ManageSoft\Launcher\CurrentVersion\TrustedLocations\<serverkey>`
	For excluded locations:
	`[Registry]\ManageSoft\Launcher\CurrentVersion\ExcludedLocations\<serverkey>`

PreferenceUpdatePeriod

Command line | Registry

`PreferenceUpdatePeriod` specifies in seconds how often the application usage component will refresh its settings from the registry. The value must be greater than 0; otherwise the default value of once every 24 hours will be used.

Values

Values / range	Integer greater than 0 (in seconds)
Default value	`86400`
Example values	`120`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o PreferenceUpdatePeriod="3600"`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

PreferIPVersion

Command line | Registry

`PreferIPVersion` determines whether network communications between components of FlexNet Manager Suite give priority to either IPv4 or IPv6 when both protocol formats are available (for example, in a dual-stack server). If the preferred format is not available, operations fail over to using the IP format that is available.

Values

Values / range

A case-insensitive string containing one of

- `IPv4`
- `IPv6`
- `SystemSupplied`. This setting means that, from the list of possible IP addresses supplied by a Dynamic Name Server (DNS), and perhaps reordered by the operating system to suit local configuration, the FlexNet tool uses the first IP address, in whichever IP version it is supplied.

Tip: *Any unrecognized value for `PreferIPVersion` invokes the same default behavior.*

Default value

```
SystemSupplied
```

This default applies when `PreferIPVersion` is not specified, or when the value is incorrectly specified and cannot be interpreted.

Example value

```
IPv6
```

Command line

Tools	Inventory agent (`ndtrack`), launcher (`ndlaunch`), or upload component (`ndupload`).
Example	`-o PreferIPVersion=IPv6`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Common`

PrioritizeRevocationChecks

Command line | Registry

Tip: *`PrioritizeRevocationChecks` is supported only on UNIX-like platforms. On Windows platforms, revocation checking behavior is determined by Group Policy. For further details, see https://technet.microsoft.com/en-us/library/ee619754%28v=ws.10%29.aspx.*

On UNIX-like platforms, `PrioritizeRevocationChecks` determines the ordering of processes for checking revocation of PKI certificates, such as certificates normally issued as part of data transfers using the HTTPS protocol. (This preferences applies only when `CheckServerCertificate` and `CheckCertificateRevocation` are both true.) Two methods are supported for checking whether a certificate has been revoked:

- Certificate Revocation Lists (CRL), which require the client device to download a file listing all certificates revoked by the relevant Certification Authority
- Online Certificate Status Protocol (OCSP), where the client device receives a response specific to the single certificate being checked.

The agent component(s) stop checking as soon as an authoritative revocation result (either affirmative or negative) is determined. For example, with the default value, if the networked OCSP check shows that the certificate is revoked, the CRL is not downloaded or checked.

Omitting one of the values from the string turns off that method of checking. For example, a command line parameter `-o PrioritizeRevocationChecks="OCSP"` limits checking to OCSP, and prevents download or checking of the CRL.

Tip: *Beware of turning off a type of checking which may be uniquely specified in the server certificate (or any intermediate certificate up the chain). For example, if a certificate specifies a URI only for CRL checking, and you use this preference as `-o PrioritizeRevocationChecks="OCSP"`, then every certificate check must by definition fail because of these contradictory settings. Recommended general practice is to use the default value, which uses the most efficient check first but fails over to the older technology if OCSP checking is not available for a certificate. Vary the setting only if your enterprise has an internal certificate authority and you are sure of the revocation settings for your internal certificates.*

A null (or unrecognized) value is the same as not having the preference set in the registry: the default value is used in these cases.

Assuming that preferences allow revocation checking, the processing order for revocation checking is:

1. The CRL cache is checked. (This is always first, regardless of the setting of `PrioritizeRevocationChecks`.)
2. If the value of `PrioritizeRevocationChecks` includes `OCSP` (whether first or second), the OCSP cache is checked.

3. When neither cache provides an authoritative result, networked resources are accessed in the order specified by `PrioritizeRevocationChecks`.
4. When neither of the networked revocation checks provides an authoritative result, the check results in a hard failure, and data transfer between the client component and the inventory beacon cannot proceed using the HTTPS protocol.

This order may result in CRL checking being used longer than expected. An example scenario might be:

1. `CheckServerCertificate=True`, `CheckCertificateRevocation=True`, `PrioritizeRevocationChecks=OCSP,CRL`. OCSP checking is on and given top priority.
2. An HTTPS transfer is attempted, and for networking reasons the OCSP server is temporarily unavailable. Accordingly, fail-over occurs, so that the CRL is downloaded for the revocation check. Since it includes a `nextUpdate` setting a week in advance, the CRL is cached.
3. Later (but within the week), another HTTPS transfer is attempted. Because there is a valid CRL in the cache, this is checked first (and provides the revocation result). A networked OCSP revocation check does not occur (despite the preference in `PrioritizeRevocationChecks`) until after the cached CRL expires.

Values

Values / range	A comma-separated list of two string literals, `OCSP` and `CRL`, in your chosen order.
Default value	`OCSP,CRL`
Example values	`OCSP`

Command line

Tool	Inventory component (`ndtrack`), installation component (`ndlaunch`), and upload component (`ndupload`)
Example	`-o PrioritizeRevocationChecks="CRL,OCSP"`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Common` or `[Registry]\ManageSoft\<Component>\CurrentVersion` where *<Component>* is the registry key for an individual component (`Tracker`, `Launcher`, or `Uploader`)

Priority

Registry

`Priority` determines the order in which upload or download connections to inventory beacons should be attempted if `AutoPriority` is `False`. (If `AutoPriority` is `True`, the `Priority` registry key is set dynamically for each download and upload activity.) The value is set independently for uploads and downloads for each inventory beacon recorded in the `[Registry]` on the managed device (that is, in the registry on Windows devices and in `config.ini` on UNIX-like platforms).

To define a location as primary server for download and/or upload, set `AutoPriority` to `False`, and set `Priority` to 1.

To define a location as one that should not be used for download and/or upload, set `AutoPriority` to `False`, and set `Priority` to `100`.

Values

Values / range	Recommended range of 0 - 100
Default value	`50`
Example values	`75`

Registry

Installed by	Failover list for inventory beacons, or manual configuration

Computer preference	For uploads:
	`[Registry]\ManageSoft\Common\UploadSettings\<reporting_location>`
	For downloads:
	`[Registry]\ManageSoft\Common\DownloadSettings\<distribution_location>`

Priority (manual mapper)

Registry

`Priority` resolves conflicting assignments of an executable (or directory) to an application for which usage is being tracked using the manual mapper.

For application usage tracking, only one application can “own” a file or directory. Where more than one application being tracked specifies the same file or directory, the value of the manual mapper `Priority` preference is used to resolve which application the file or directory is assigned to for tracking. This resolution process occurs at the time the application is being specified for tracking: the application with the highest value for `Priority` owns the file or directory for usage tracking. Where more than one application specifies the same file or directory for usage tracking, and both have identical priorities, the application whose usage tracking is most recently defined takes precedence.

Where no specific `Priority` is assigned, the value of `ManualMapperDefaultPriority` is silently assigned (see ManualMapperDefaultPriority).

This `Priority` preference is required only when the manual mapper is used to identify files to monitor (and is otherwise ignored). This registry value must be created manually, within a node for the chosen application that has also been inserted manually (and shown below as *`Application node`*).

Values

Values / range	Integer between 1 and 10000.
Default value	Value set for `ManualMapperDefaultPriority`.
Example value	`1000`

Registry

Installed by	Manual configuration only.
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion\Manual Mapper\`*`Application node`*

ProcessUpdatePeriod

Command line | Registry

`ProcessUpdatePeriod` specifies in seconds how often the application usage component checks for newly started or exited applications. The value must be greater than 0; otherwise the default value will be used.

Values

Values / range	Integer greater than 0 (in seconds)
Default value	`60`
Example values	`120`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o ProcessUpdatePeriod="90"`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

ProductUpdatePeriod

Command line | Registry

`ProductUpdatePeriod` specifies in seconds how often the application usage agent will check for newly installed applications. The value must be greater than 0; otherwise the default value will be used.

Values

Values / range	Integer greater than 0 (in seconds)
Default value	`86400`
Example values	`3600`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o ProductUpdatePeriod="90"`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

ProgramFiles, ProgramFilesX86Folder, ProgramFilesX64Folder

Command line | Registry

`ProgramFiles, ProgramFilesX86Folder, ProgramFilesX64Folder` are a set of options that point to the Windows program files folder (across various versions of the operating system) on the target device where inventory is being gathered. `Program Files` exists for backwards compatibility; current distributions operate using `ProgramFilesX86Folder`. These preferences are ignored on UNIX-like platforms.

Note: *`ndtrack` (the executable underlying both the FlexNet inventory agent and FlexNet Inventory Scanner) is a 32-bit executable.*

Values

Values / range	The valid folder name where applications are installed.
Default values	Default values respectively on a 32-bit platform are: • `Program Files` • `Program Files` • "" (empty value). Default values on a 64-bit platform are: • `Program Files` • `Program Files (x86)` • `Program Files`
Example values	`-o ProgramFiles=”D:\Program Files”`

Command line

Tool	Inventory component (`ndtrack`)
Example	`option`

Registry

Installed by	Predefined by Microsoft Windows.

Reference as	`$(ProgramFiles)`

Protocol

Registry

`Protocol` identifies the uploads (or download) protocol for transferring files from (or to) the managed device.

Values

Values / range	`http`, `https`, `file`, `ftp` ***Note:*** *Only `http` and `https` are supported by inventory beacons. The other values are available for testing or custom solutions.*
Default value	(No default.)

Registry

Installed by	Failover list for inventory beacons, or manual configuration

Computer preference	For uploads:
	`[Registry]\ManageSoft\Common\UploadSettings\<reporting_location>`
	For downloads:
	`[Registry]\ManageSoft\Common\DownloadSettings\<distribution_location>`
	For trusted locations:
	`[Registry]\ManageSoft\Launcher\CurrentVersion\TrustedLocations\<serverkey>`
	For excluded locations:
	`[Registry]\ManageSoft\Launcher\CurrentVersion\ExcludedLocations\<serverkey>`

Recurse

Command line | Registry

`Recurse` controls whether the tracker (`ndtrack` executable) drills down for inventory collection:

- When set to `True`, the tracker includes folders beneath the top-level folder(s) specified by `IncludeDirectory` or `EmbedFileContentDirectory`.
- When set to `False`, the tracker does not recurse folders beneath the top level folder(s). It only tracks files immediately within the folder(s) specified by `IncludeDirectory` or `EmbedFileContentDirectory`.

Values

Values / range	Boolean (`True` or `False`).
Default value	`True`
Example values	`False`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o Recurse=False`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

Regex

Registry

`Regex` is a Boolean that determines whether `ExecutablePath` is taken as a string literal (`False`) or processed as a regular expression (`True`). This `Regex` preference is required only when the manual mapper is used to identify files to monitor (and is otherwise ignored). This registry key must be created manually, within a node for the chosen application that has also been inserted manually (and shown below as *`Application node`*). For more information, see ExecutablePath.

Some typical examples of special characters that may be used within `ExecutablePath` when `Regex=True` include:

Regular expression	Matches
`.` (the period character)	Any single character. (Because of this special purpose, within a path using a regex, the period separator between file name and file extension should be escaped.)
* (asterisk)	Matches the preceding character one or more times.
`[xyz]` (square brackets around a set of characters)	Matches any one of the enclosed characters in the set. A range of characters can be specified using a hyphen: for example, `[a-d]` is the same as `[abcd]`. The square brackets may also be used to terminate (or group) other parts of the regex, such as the next example. Finally, the square brackets may be used to escape characters that usually have special meaning, so that they are interpreted as a normal string – for example, within a regex `[.]` means the normal period separator between a file name and file extension.
`x|y` (pipe character between alternatives)	Matches x or y. For example, `Office [10|11]` matches `Office 10` or `Office 11`, but not Office 12.

Values

Values / range	Boolean (`True` or `False`)
Default value	No default value in the registry. When not specified, behavior defaults to `False`, so that `ExecutablePath` is taken as a string literal.
Example values	`True`

Registry

Installed by	Manual configuration only.
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion\Manual Mapper\`*`Application node`*

RunInventoryScripts

Command line | Registry

If `RunInventoryScripts` is `True`, then at the completion of inventory gathering on Windows target devices, any custom `.vbs` scripts saved in the location specified by `InventoryScriptsDir` are executed. Only `.vbs` scripts are executed: any other scripts saved in the same location are ignored. By default, no custom scripts are run.

Tip: *This preference is set to `true` by `InventorySettings.xml`, which is required for advanced inventory gathering for CALs, Microsoft Exchange, and so on. If you have licensed the FlexNet Manager for Datacenters product, `InventorySettings.xml` is in use, and the effective default value is reversed to `true`.*

This preference is ignored for UNIX-like platforms.

Values

Values / range	Boolean (`True` or `False`).
Default value	`False` (May be changed by `InventorySettings.xml`.)
Example values	`True`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o RunInventoryScripts=True`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

ScheduleType

Command line only

`ScheduleType` applies only for Windows managed devices. Determines the type of schedule that is run when the scheduling agent is run via the command line. There are two options:

- `Machine`—Runs the schedule agent in “Machine” mode. You must have administrator privileges, or run under the Local System account. Any schedules installed on the same command line are installed as Machine schedules. This option can also be used to view scheduled tasks from the currently installed Machine schedule. (A Machine schedule is run for the machine on which it is installed, regardless of any users that may or may not have accounts on that machine.)
- `User`—Runs the scheduling agent in “User” mode. Will not work if the scheduling agent is run as the Local System account. Any schedules installed on the same command line are installed as User schedules. This option can also be used to view scheduled tasks from the currently installed User schedule. (A User schedule is run for the specified user account on the machine where the schedule resides.)

Values

Values / range	`Machine`, `User`

Default value	If running as the Local System account:
	`Machine`
	If running as any User account (including Administrator):
	`User`

Command line

Tool	Scheduling component (`ndschedag`)
Example	`-o ScheduleType=User`

ScriptDir

Registry

`ScriptDir` identifies the starting directory for a tree where scripts used by the FlexNet inventory agent are stored. It is referenced by other preferences.

Values

Values / range	A string identifying an existing directory on (or accessible by) the managed device.
Default value	`$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Scripts`
Example values	`\\my-organization-server\Scripts\`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Common`

SelectorAlgorithm

Registry

`SelectorAlgorithm` specifies the algorithm(s) used to assign values to the `Priority` registry key for download and upload locations.

After application of the nominated algorithm(s), the managed device will attempt to collect packages from the highest priority server. In the event of connection failure, the managed device uses the other prioritized servers remaining in the list as failover servers.

Tip: *Failover inventory beacons must be configured for anonymous authentication.*

The NetSelector includes the following algorithms:

- `MgsADSiteMatch`: Moves all servers in the current managed device's site to the front of the priority list (supported only on Microsoft Windows inventory devices)
- `MgsBandwidth`: Priorities are based on end-to-end bandwidth availability to the server
- `MgsDHCP`: Priorities are based on lists of servers specified in DHCP (supported only on Microsoft Windows inventory devices)
- `MgsDomainMatch`: Priorities are determined by closest match in domain name
- `MgsIPMatch`: Priorities are determined by closest IP address match
- `MgsNameMatch`: Matches prefixes in computer names
- `MgsPing`: Priorities are determined by fastest ping response time
- `MgsRandom`: Random priorities are assigned
- `MgsServersFromAD`: Priorities are determined according to lists of servers specified in Active Directory (supported only on Microsoft Windows inventory devices)
- `MgsSubnetMatch`: Moves all servers in the current subnet to the front of the priority list, but retaining the relative order of existing priorities.

Each algorithm may be given an integer parameter that determines the number of servers to which priorities will be assigned. Some algorithms may also be given an additional boolean attribute that can cause unmatched servers to be discarded from the list (priority set to the string literal invalid). Some algorithms also accept other parameters.

Values

Values / range	`MgsADSiteMatch`, `MgsBandwidth`, `MgsDHCP`, `MgsDomainMatch`, `MgsIPMatch`, `MgsNameMatch`, `MgsPing`, `MgsRandom`, `MgsServersFromAD`, `MgsSubnetMatch`
Default value	`MgsRandom;MgsPing;MgsADSiteMatch`

Example values	`MgsRandom(3)`
	This means that ManageSoft should randomly assign the top three servers (based on the priorities currently assigned).
	`MgsADSiteMatch(, True);MgsSubnetMatch`
	This means that the NetSelector lists servers outside the current managed device's site as "invalid". (`MgsSubnetMatch` will only prioritize valid servers set by `MgsADSiteMatch`.)

Registry

Installed by	Installation of FlexNet inventory agent
Computer preference	`[Registry]\ManageSoft\NetSelector\CurrentVersion`

SendTCPKeepAlive

Command line | Registry

`SendTCPKeepAlive` determines whether the uploader (both in the Windows-based FlexNet inventory agent and in FlexNet Beacon) sends keep-alive TCP (socket level) packets to the destination server at 3-minute intervals. This behavior is helpful where you have large files (10MB or larger) that take several minutes to resolve into the inventory database in the central application server (or inventory server, in a larger, multi-server implementation). This may happen, for example, with large `.ndi` inventory files, large Active Directory imports, or the like. When this is the case and there are *no* keep-alive packets sent, the intermediate network infrastructure may decide that the connection is idle, and terminate it. As a result, even though the inventory/import is resolved and processed entirely normally, the disconnection means that the inventory beacon cannot be notified of the success. It then follows normal protocol for a failed upload, saving the source file locally on the inventory beacon, and re-trying the upload. The result is additional load on the network and on the central application server as the repeated, unchanged uploads are unnecessarily resolved.

Tip: *When the Windows-based FlexNet inventory agent is uploading to a stand-alone inventory beacon, keep-alive TCP packets are less likely to be needed, since the inventory beacon quickly saves the uploaded file, ready for a repeated upload to the next destination in the hierarchy. Lengthy delays are unlikely. However, if your inventory beacon is co-installed on [one of] your central application server[s], it does not normally save files, but automatically hands the files off to the resolvers for loading directly into the appropriate database (which is where the delays may occur). Therefore, if you have instances of FlexNet inventory agent uploading to an inventory beacon that is co-located on your application server, it is strongly recommended that you leave the default* `true` *value for* `SendTCPKeepAlive`*.*

These problems are avoided when the uploader sends keep-alive packets on the connection, and can therefore receive notification of the successful resolving of the import. For this reason, the default behavior is for the uploader to send socket-level keep-alive packets.

Tip: *Remember that the FlexNet inventory agent makes a single attempt to upload immediately after collecting inventory. In that context, if this setting is false so that a large upload cannot be flagged as successfully completed, the repeated uploads happen in the catch-up period when the uploader is triggered separately (typically after hours). However, the* `SendTCPKeepAlive` *setting is used by code common to the uploader and to the tracker (inventory component), so that the behavior is identical for all uploads. Therefore when* `SendTCPKeepAlive` *is true (the default), the original upload by the tracker is more likely to succeed, regardless of inventory file size.*

The uploader may terminate the connection under either of these scenarios:

- **Connection failure:** If the destination server does not respond to a keep-alive packet within 10 seconds, the keep-alive request is repeated 10 times at 10 second intervals. If there is still no response, the uploader closes the connection and logs an upload failure. (For a separate and overriding case of network failure, see `NetworkTimeout`.)
- **All files of this type resolved:** If the destination server sends the uploader a success message that the last uploaded file (say, an `.ndi` inventory file) has been resolved, the uploader takes either of two paths:
  - If it has further files of the same type (in this example, more `.ndi` files) awaiting upload, it resets the keep-alive timer and tries to reuse the same connection to upload the next file of the same type. (At this point, the `MaxKeepAliveLifetime` or `MaxKeepAliveRequests` settings may refuse reuse of the same connection, and a new connection is requested.)
  - When there are no more files of the same type awaiting upload, the uploader terminates the connection. If there are additional files of *different* types awaiting upload, it requests a *new* connection for each file type (such as `.disco` files, Active Directory imports, log files, and the like).

This preference is ignored on UNIX-like platforms.

Values

Values / range	Boolean (`True`/`False`)
Default value	`True`
Example values	`False`

Command line

Tool	Uploader component (`ndupload`)
Example	`-o SendTCPKeepAlive=false`

Registry

Installed by	Installer, or manual configuration
Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion` `[Registry]\ManageSoft\Common`

SessionBackupPeriod

Command line | Registry

`SessionBackupPeriod` specifies in seconds how often the application usage agent caches application usage data that is currently saved in memory. The value must be greater than 0; otherwise the default value will be used.

Values

Values / range	Integer greater than 0 (in seconds)
Default value	`3600`
Example values	`900`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o SessionBackupPeriod=90`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

ShowIcon (installation component)

Command line | Registry

`ShowIcon` influences visibility to the user on the computer device being inventoried:

- When set to `True`, `ndlauch` displays an icon in the system tray when it is installing a package. This icon displays,

regardless of the value of the UserInteractionLevel (installation agent) preference. If this icon is double-clicked and UserInteractionLevel (installation agent) is set to Status or Auto, the progress display toggles from being hidden to being visible.

- When set to False, no icon will display.

Tip: *On modern versions of Windows, this preference affects FlexNet inventory agent when it is being executed by a particular user account. When (as is normal) the inventory is being run by the local SYSTEM account, no icon is visible in the system tray.*

Values

Values / range	Boolean (True or False).
Default value	No registry default. If no value is supplied, the behavior depends on UserInteractionLevel. Where this is Full, the behavior is equivalent to ShowIcon=False; and otherwise the behavior is as for ShowIcon=True.
Example values	`False`

Command line

Tool	Installation component (ndlaunch)
Example	`-o ShowIcon=False`

Registry

Installed by	Installation of FlexNet inventory agent
Computer preference	In order of precedence: • [Registry]\ManageSoft\Launcher\CurrentVersion • [Registry]\ManageSoft\Common

ShowIcon (inventory component)

Command line | Registry

ShowIcon influences visibility to the user on the Windows computer device being inventoried by a non-SYSTEM user account:

- When set to True, the tracker (ndtrack.exe) displays an icon in the system tray when it is collecting inventory. This

icon displays, regardless of the value of the `UserInteractionLevel` preference. If this icon is double-clicked and `UserInteractionLevel` is set to `Status` or `Auto`, the progress display toggles from being hidden to being visible.

- When set to `False`, no icon will display.

Tip: *On modern versions of Windows, this preference affects the tracker when it is being executed by a particular user account. When (as is normal) the inventory is being run by the local SYSTEM account, no icon is visible in the system tray.*

Values

Values / range	Boolean (`True` or `False`).
Default value	No registry default. If no value is supplied, the behavior depends on `UserInteractionLevel`. Where this is `Full`, the behavior is equivalent to `ShowIcon=False`; and otherwise the behavior is as for `ShowIcon=True`.
Example values	`False`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o ShowIcon=False`

Registry

Installed by	Installation of FlexNet inventory agent
Computer preference	In order of precedence: • `[Registry]\ManageSoft\Tracker\CurrentVersion` • `[Registry]\ManageSoft\Common`

Software

Command line | Registry

`Software` is an internal setting and not for general use as a preference. It is set to true whenever any of the following is true:

- IncludeRegistryKey

- ManageSoftPackages
- MSI
- PlatformSpecificPackages.

It is then checked before the tracker performs local Oracle inventory gathering, which is disabled if Hardware is true and `Software` is false.

Somewhat redundantly but for safety, this preference is also set to false by the tracker command line used for high-frequency hardware inventory checks, as required for subcapacity calculations for IBM PVU licenses.

Values

Values / range	Boolean (`true`/`false`)
Default value	None.
Example values	Not applicable.

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o Software=false`

Registry

Installed by	Code internals
Computer preference	Not applicable

SourceFile

Command line | Registry

`SourceFile` identifies the file or files to be uploaded by the upload agent.

Values

Values / range	Either a UNC (`\\MYCOMPUTER\...`) or a drive (`C:\`) path to the required file or files. Wildcard characters can be used in the filename component.
Default value	(No default.)

Example values	`C:\Temp\*.log`

Command line

Tool	Upload component (`ndupload`)
Example	`-o SourceFile=c:\temp\mylogfile.log`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion`

SourceRemove

Command line | Registry

`SourceRemove` determines whether the upload agent removes the uploaded file(s) from the source location after a successful upload. If `True`, the files are removed from the source location.

Values

Values / range	Boolean (`True` or `False`)
Default value	`True`
Example values	`False`

Command line

Tool	Upload component (`ndupload`)
Example	`-o SourceRemove` `-o SourceRemove=False`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion`

SSLCACertificateFile

Command line | Registry

`SSLCACertificateFile`, available only for UNIX-like platforms, gives the path and file name for the file that concatenates all trusted certificates in the chain to the Certificate Authority. For details about the file format, see Agent Third-Party Deployment: HTTPS CA Certificate File Format (UNIX).

You can combine this file with another storage option that allows multiple separate certificate files to be stored in a directory (see SSLCACertificatePath), in which case the agent(s) scan the combined file first, and then check the certificates in the folder.

Values

Values / range	A valid file path (or variable that references the file path) and file name.
Default value	`$(SSLDirectory)/cert.pem` The default expansion is `/var/opt/managesoft/etc/ssl/cert.pem`.
Example values	`/tmp/test/cert.pem`

Command line

Tool	Inventory component (`ndtrack`), installation component (`ndlaunch`), and upload component (`ndupload`)
Example	`-o SSLCACertificateFile="/tmp/test/cert.pem"`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Common` or `[Registry]\ManageSoft\<Agent>\CurrentVersion` where *<Agent>* is the registry key for an individual component (`Tracker`, `Launcher`, or `Uploader`)

SSLCACertificatePath

Command line | Registry

`SSLCACertificatePath`, supported only on UNIX-like platforms, gives a directory path in which multiple trusted Certification Authority certificates may be saved.

***Tip:** Individual certificate files must be named with the CA subject name hash value (such as `9d66eef0`).*

You can combine the use of this folder with a merged certificate file (see SSLCACertificateFile), in which case the file is searched first, and then the directory of individual certificates.

Values

Values / range	Any valid path and directory existing on the client device.
Default value	`$(SSLDirectory)/certs` The default expansion is `/var/opt/managesoft/etc/ssl/certs`
Example values	`/tmp/test/certs`

Command line

Tool	Inventory component (`ndtrack`), installation component (`ndlaunch`), and upload component (`ndupload`)
Example	`-o SSLCACertificatePath="/tmp/test/certs"`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Common` or `[Registry]\ManageSoft\<Component>\CurrentVersion` where *<Component>* is the registry key for an individual component (`Tracker`, `Launcher`, or `Uploader`)

SSLClientCertificateFile

Command line | Registry

For authentication when using mutual Transport Layer Security (TLS) to secure HTTPS communications, the server (or inventory beacon) asks the client (the managed inventory device) for a valid certificate that must be verified before the

authentication process can complete. This means that the inventory device must have a locally-stored certificate available in PEM format. The `SSLClientCertificateFile` preference, available only for UNIX-like platforms, gives the path and file name for that certificate file on the inventory device. (For comparison, Windows devices have all required certificates saved in the certificate store, in the registry under the `HKEY_LOCAL_MACHINE` root.)

Note: *Unlike SSLCACertificateFile, which records an entire chain of certificates, this preference records just the one client certificate to be used for mutual TLS. This certificate must be recorded here separately, and not included with any other certificates listed in `SSLCACertificateFile`. Furthermore, this special certificate for mutual TLS has its own distinct path, and does not share in SSLCACertificatePath.*

Values

Values / range	A valid file path (or variable that references the file path) and file name.

Default value	`$(SSLDirectory)/client/client_cert.pem`
	If this default is not available in the pseudo-registry, it is supplied from code internals.
	 Tip: *The appropriate environment variables are saved in the pseudo-registry in `/var/opt/managesoft/etc/config.ini`, which is created when the FlexNet inventory agent is installed. (For updates to this file, see Agent Third-Party Deployment: Updating config.ini on a UNIX Device.) The default expansion for the environment variable `$(SSLDirectory)` is*
	`$(CommonAppDataFolder)/etc/ssl`
	In turn, the default expansion for `$(CommonAppDataFolder)` is
	`/var/opt/managesoft`
	Therefore the fully-expanded path and file name defaults to
	`/var/opt/managesoft/etc/ssl/client/client_cert.pem`
	The client private key is saved in a `private` subdirectory, so that its default location is
	`/var/opt/managesoft/etc/ssl/client/private`

Example values	`/tmp/test/client_cert.pem`

Command line

Tool	Installation component (`ndlaunch`), inventory component (`ndtrack`), upload component (`ndupload`)

Example	`-o SSLCACertificateFile="/tmp/test/cert.pem"`

Registry

Installed by	Installation, or manual configuration of the `config.ini` file

Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`
	`[Registry]\ManageSoft\Tracker\CurrentVersion`
	`[Registry]\ManageSoft\Uploader\CurrentVersion`
	`[Registry]\ManageSoft\Common`

SSLClientPrivateKeyFile

Command line | Registry

For mutual Transport Layer Security (TLS), the client (the managed inventory device) requires a private key. The private key is never sent to the server (the inventory beacon), but is used to sign a piece of data sent to the server, where that data verifies that the client certificate (see SSLClientCertificateFile) was issued to your enterprise, which owns the corresponding private key. Thereafter the server uses the certificate (which was signed by a Certificate Authority that is trusted by the server) to decrypt data transfers.

For UNIX-like platforms only, the `SSLClientPrivateKeyFile` preference gives the path and file name for the file that contains the private key in PEM format.

Values

Values / range	A valid file path (or variable that references the file path) and file name.
Default value	`$(SSLDirectory)/client/private/client_key.pem` The default expansion for the environment variable gives this path and file name: `/var/opt/managesoft/etc/ssl/client/private/client_key.pem`
Example values	`/tmp/test/client_key.pem`

Command line

Tool	Installation component (`ndlaunch`), inventory component (`ndtrack`), upload component (`ndupload`)
Example	`-o SSLClientPrivateKeyFile="/tmp/test/client_key.pem"`

Registry

Installed by	Code internals, or manual configuration

Computer preference	`[Registry]\ManageSoft\Launcher\CurrentVersion`
	`[Registry]\ManageSoft\Tracker\CurrentVersion`
	`[Registry]\ManageSoft\Uploader\CurrentVersion`
	`[Registry]\ManageSoft\Common`

SSLCRLCacheLifetime

Command line | Registry

`SSLCRLCacheLifetime`, supported only for UNIX-like platforms, sets the maximum lifetime of certificate Revocation Lists (CRLs) cached in the `SSLCRLPath`, expressed as a whole number of seconds. A cached CRL is expired on the earlier of:

- Its own `nextUpdate` value (which is the certificate's *valid until* date), or
- The sum of the `SSLOCSPCacheLifetime` and the operating system's `Last modified` date/time on the cached file.

The special case of `0` means that the cache lifetime is disabled, and a CRL expires as set in its `nextUpdate` field. (If the CRL does not have any `nextUpdate` value when the `SSLCRLCacheLifetime=0`, the CRL is not cached.)

Depending on your environment, one possible use is to set this to about 10 minutes (600 seconds). This is sufficient for an agent to complete a policy update, for example, and then refresh the cache on the next occurrence.

Values

Values / range	Zero, or a positive integer.
Default value	`0` This default means that certificate validity period is as specified on the certificate itself.
Example values	`600`

Command line

Tool	Inventory component (`ndtrack`), installation component (`ndlaunch`), and upload component (`ndupload`)
Example	`-o SSLCRLCacheLifetime=600`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Common` or `[Registry]\ManageSoft\<Component>\CurrentVersion` where *<Component>* is the registry key for an individual component (`Tracker`, `Launcher`, or `Uploader`)

SSLCRLPath

Command line | Registry

`SSLCRLPath`, supported only on UNIX-like platforms, identifies a directory that stores cached certificate revocation lists.

Values

Values / range	Any valid path and directory name.
Default value	`$(SSLDirectory)/crls` The default expansion is `/var/opt/managesoft/etc/ssl/crls`
Example values	`/tmp/test/crl-cache`

Command line

Tool	Inventory component (`ndtrack`), installation component (`ndlaunch`), and upload component (`ndupload`)
Example	`-o SSLCRLPath="/tmp/test/crl-cache"`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Common` or `[Registry]\ManageSoft\<Component>\CurrentVersion` where *<Component>* is the registry key for an individual component (`Tracker`, `Launcher`, or `Uploader`)

SSLDirectory

Command line | Registry

`SSLDirectory`, supported only on UNIX-like platforms, defines a base directory for folders and files related to Transport Layer Security (TLS). Its value is then referenced in other settings.

You can set this as a common 'registry' entry (in `/var/opt/managesoft/etc/config.ini`), so that the same behavior occurs across all relevant components; and you can override the common behavior by setting an overriding entry for any individual component if required.

Values

Values / range	Any valid directory path.
Default value	`$(CommonAppDataFolder)/etc/ssl` By default, this expands to `/var/opt/managesoft/etc/ssl`.
Example values	`/tmp/test`

Command line

Tool	Inventory component (`ndtrack`), installation component (`ndlaunch`), and upload component (`ndupload`)
Example	`-o SSLDirectory="/tmp/test"`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Common` or `[Registry]\ManageSoft\`*`<Component>`*`\CurrentVersion` where *`<Component>`* is the registry key for an individual component (`Tracker`, `Launcher`, or `Uploader`)

SSLOCSPCacheLifetime

Command line | Registry

`SSLOCSPCacheLifetime`, supported only for UNIX-like platforms, sets the maximum lifetime of OCSP responses cached in the `SSLOCSPPath`, expressed as a whole number of seconds. A cached response is expired on the earlier of:

- Its own `nextUpdate` value (which is the certificate's *valid until* date), or
- The sum of the `SSLOCSPCacheLifetime` and the operating system's `Last modified` date/time on the cached file.

The special value of `0` means that the cache lifetime is disabled, and an OCSP response expires as set in its `nextUpdate` field. (If the OCSP response does not have any `nextUpdate` value when the `SSLOCSPCacheLifetime=0`, the response is not cached.)

Depending on your environment, one possible use is to set this to about 10 minutes (600 seconds). This is sufficient for an agent to complete a policy update, for example, and then refresh the cache on the next occurrence.

Values

Values / range	Zero, or a positive integer.
Default value	`0` This default means that certificate validity period is as specified on the certificate itself.
Example values	`600`

Command line

Tool	Inventory component (`ndtrack`), installation component (`ndlaunch`), and upload component (`ndupload`)
Example	`-o SSLOCSPCacheLifetime=600`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Common` or `[Registry]\ManageSoft\<Component>\CurrentVersion` where *<Component>* is the registry key for an individual component (`Tracker`, `Launcher`, or `Uploader`)

SSLOCSPPath

Command line | Registry

`SSLOCSPPath`, supported only on UNIX-like platforms, identifies a directory that stores responses to OCSP (Online Certificate Status Protocol) checks of revocation status on server PKI certificates normally issued as part of HTTPS data transfers.

Values

Values / range	Any valid path and directory name.
Default value	`$(SSLDirectory)/ocsp` The default expansion is `/var/opt/managesoft/etc/ssl/ocsp`
Example values	`/tmp/test/ocsp-cache`

Command line

Tool	Inventory component (`ndtrack`), installation component (`ndlaunch`), and upload component (`nduploa`d)
Example	`-o SSLCRLPath="/tmp/test/ocsp-cache"`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Common` or `[Registry]\ManageSoft\<Component>\CurrentVersion` where *<Component>* is the registry key for an individual component (`Tracker`, `Launcher`, or `Uploader`)

Startup

Command line | Registry

Warning: *Internal use only: do not edit.*

The scheduling agent checks the `Startup` flag to determine if it has been called as part of system startup or while an end-user is logging on:

- For system startup, the scheduling agent is run by the scheduled event `MGSStartup`.
- For logon processing (Windows only), the scheduling agent is run as a result of the value for the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows \CurrentVersion\Run\SchedulingAgent_nDG`. The scheduling agent cannot be called during logon processing for non-Windows devices.

The command line parameter `-o Startup=True` is passed in by default. Setting this preference elsewhere in the registry will have no effect.

Notice: *If the command line is changed to `-o Startup=False`, the scheduling agent will still trigger missed events*

if configured.

Values

Values / range	Boolean (`True` or `False`)
Default value	`False`

Command line

Tool	Scheduling component (`ndschedag`)
Example	`-o Startup=True`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)
Computer preference	`[Registry]\ManageSoft\Schedule Agent\CurrentVersion`

StrictInstall

Command line | Registry

If `StrictInstall` is set to `True`, the policy agent returns a non-zero exit code if any package in policy fails to install. If set to `False`, the policy agent may return a zero exit code even if packages failed to install. Do not use the policy agent's return code to test for success unless this preference is set to `True`.

Values

Values / range	Boolean (`True` or `False`)
Default value	No registry default; default behavior False
Example values	`True`

Command line

Tool	Policy component (`mgspolicy`), installation component (`ndlaunch`)

Example	`-o StrictInstall=True`

Registry

Installed by	Manual configuration
Computer preference	In order of precedence: • `[Registry]\ManageSoft\Launcher\CurrentVersion` • `[Registry]\ManageSoft\Common`

SysDirectory

Command line | Registry

`SysDirectory` is a FlexNet pre-defined variable for the path to the Windows `System` folder. Intended for use as a reference, as `$(SysDirectory)`, but the value can be over-written.

Values

Values / range	Valid folder name.
Default value	The path to the system folder on the Windows operating system.
Example values	`C:\Winnt\System32`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o SysDirectory=C:\Windows\System`

Registry

Installed by	Predefined within FlexNet inventory agent / Windows.
Reference as	`$(SysDirectory)`

TrackProductKey

Command line | Registry

`TrackProductKey` determines whether the inventory component reports product keys from Microsoft Installer (MSI) packages as part of software inventory. Normal behavior is to include the MSI keys; but this preference is set false in the tracker command line used for high-frequency hardware inventory checks to support subcapacity calculations for IBM PVU licenses.

Values

Values / range	Boolean (`true` or `false`)
Default value	`true` This is the default behavior when the preference is not specified.
Example values	`false`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o TrackProductKey=false`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

UILevel

Command line | Registry

`UILevel` is a synonym (or alias) for UserInteractionLevel (installation component), and is somewhat faster to type in a command line.

UIMode

Command line only

UIMode only applies for Windows managed devices.

Determines the user interface when the scheduling agent is run from the command line. There are two options:

- LogUI—Display the event log user interface.
- EventUI—Display currently scheduled events.

Values

Values / range	LogUI, EventUI
Default value	EventUI

Command line

Tool	Scheduling component (ndschedag)
Example	-o UIMode=LogUI

Upload

Command line | Registry

When Upload is set to True, the inventory files generated by the inventory tool are immediately uploaded to the reporting location on the inventory beacon. When set to False, the inventory files are not uploaded (for example, in case you wish to inspect or validate their contents).

Values

Values / range	Boolean (True or False).
Default value	There are separate defaults for different cases: • In the Adopted case or Agent third-party deployment case (when the inventory tool is locally installed on the target inventory device and receiving policy updates). the default is True. • In the FlexNet Inventory Scanner case, the default is False. • In the Zero-footprint case, the preference is not applicable.
Example values	False

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o Upload=False`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

UploadLocation

Command line | Registry

Unless the `Upload` parameter is `False` (when this preference is ignored), an upload location must be specified, and `UploadLocation` is a convenient method. The value must be a URL of the form:

```
protocol separator server-name/webService
```

where

- *`protocol`* and *`separator`* comprise one of these combinations (note three slashes for the file protocol):
  - `http://`
  - `https://`
  - `ftp://`
  - `file:///`

  Tip: *If you are uploading to an inventory beacon, use one of the HTTP or HTTPS protocols. The FlexNet Beacon software does not provide native support for FTP or file shares. (Those protocols are available only for configuration of your own uploads, such as for disconnected inventory beacons.) (In addition, on UNIX-like platforms for the scanner-like `ndtrack.sh`, network shares are not supported.)*

- *`server-name`* is either the IP address or a server name (where the system running the inventory tool has DNS access to resolve server names) of the destination for the upload (such as a parent inventory beacon, or a central application server).

  Tip: *If the target server is not listening on the default port number for the selected protocol, add the port number to the server name, separated by a colon. Example:*

  ```
  https://myServer.example.com:886/ManageSoftRL
  ```

Note: *Where you cannot use a host name or fully-qualified domain name for the server hosting the upload location, you may specify an IP address. Because this IP address is being specified from the managed device's perspective, both the IPv4 and IPv6 address families are supported. This is true across platforms (Windows platforms using the registry, or UNIX-like platforms using* `config.ini`*), as well as for both registry values and command line parameters. (For clarity, keep in mind that if you specify the same relationship from the other end, by customizing the* `BeaconEngine.conf` *file on an inventory beacon, the IPv6 address family is specifically excluded. That is because the* `BeaconEngine.conf` *value is shared through all inventory beacons to all policy-driven FlexNet inventory agents, legacy versions of which do not support the IPv6 address family. In contrast, the local command line or preference setting on a specific target device is not shared more widely.) Therefore all of the following variants are valid settings on a managed device:*

```
http://myServer.example.com/ManageSoftRL
https://203.0.113.122/ManageSoftRL
http://2001:db8::01a6/ManageSoftRL
```

- *`webService`* is the mandatory string `ManageSoftRL`, which is the name of the web service that receives the uploaded inventory, and stores it (briefly) in *`%CommonAppData%`*`\Flexera Software\Incoming\Inventories` on the inventory beacon. The scheduled task `Upload Flexera logs and inventories`, which by default runs every minute, then transfers the file to the central server.

Note: *If* `Upload` *is* `True` *and* `UploadLocation` *is not supplied, the uploader looks for upload information in* `[Registry]\ManageSoft\Common\UploadSettings` *(and similarly in* `HKEY_CURRENT_USER`*), where the upload information inside each key will be used in sequence until the file(s) are successfully uploaded. (The uploader ignores servers listed in* `UploadSettings` *whose priority values are "invalid" or non-numeric.)*

Values

Values / range	Any valid URL.
Default value	(No default.)
Example values	`ftp://server/dir1/dir2` (Such a value is not supported for upload to inventory beacons, but is available for custom upload arrangements.)

Command line

Tool	Upload component (`ndupload`) or inventory component (`ndtrack`)
Example	`-o UploadLocation=http://server/ManageSoftRL`

Registry

Installed by	Manual configuration

Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion`
	`[Registry]\ManageSoft\Tracker\CurrentVersion`
	`[Registry]\ManageSoft\Common`

UploadPassword

Command line | Registry

`UploadPassword` provides the password of the specified `UploadUser`. If the option is not used, the upload agent attempts to use the sequence of upload passwords from the registry.

Values

Values / range	A valid password.
Default value	(No default.)
Example values	`mypassword`

Command line

Tool	Upload component (`ndupload`)
Example	`-o UploadPassword=mypassword`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion`

UploadPeriod

Command line | Registry

`UploadPeriod` specifies in seconds how often the application usage component uploads recorded application usage data to a selected inventory beacon. The value must be greater than 0; otherwise the default value will be used.

Values

Values / range	Integer greater than 0 (in seconds)
Default value	`28800`
Example values	`3600`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o UploadPeriod="3600"`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

UploadRule

Command line | Registry

`UploadRule` identifies the upload rule governing this command. Each rule covers a specific file type, mapping it to an upload destination set aside for receiving the matching file type. The mappings can be found in `[Registry]\ManageSoft\Common\Rules`. Setting this option also tells the uploader to transfer files of the specified type.

Note: *If this option is set, any value of `UploadLocation` is ignored.*

Values

Values / range	One of `Inventory`, `Log`, `PolicyComplianceLog`, `SMSStatusMessage`
Default value	(No default.)
Example values	`Log`

Command line

Tool	Upload component (`ndupload`)
Example	`-o UploadRule=Log`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion`

UploadSettings

Registry

`UploadSettings` is a registry key (container) for several preferences that can control the upload of data by the FlexNet inventory agent (or the lightweight inventory scanner configurable for UNIX-like platforms, where the values can be stored in the `ndtrack.ini` configuration file). These registry values are in sets that apply to a particular reporting location, for which reason the registry key must be completed with an identifier for the reporting location. The completed path leads to the relevant set of registry values, as shown below.

When configured by the failover list generated by an inventory beacon, the placeholder *<reporting_location>* takes the form of a GUID that identifies the reporting location on the particular inventory beacon (for example, `{8909c9ba-8492-420e-b6e0-100ecf115b0a}`). In contrast, if you are manually configuring `UploadSettings` in an `ndtrack.ini` file for the lightweight inventory scanner on UNIX-like platforms, you may use any string of ASCII characters (excluding white space) that is unique within the context of the `ndtrack.ini` file for these locations.

Four name/value pairs must be specified, and others are optional. To omit an optional value, you may include the name and leave the value blank (as shown in the example below), or omit the name/value pair entirely. The values that may be set are:

- `Protocol` – Mandatory. For upload to an inventory beacon, this must be either `http` or `https`.
- `Name` – a user-friendly (general purpose) name for this group of settings. When set by policy downloaded from an inventory beacon, this consists of the value of Host with the string " `Reporting Location`" appended.
- `Directory` – Mandatory, and must be called `ManageSoftRL` (this is the name of a web service on the inventory beacon that accepts uploaded files and by default saves them to *%CommonAppData%*`\Flexera Software\Incoming\Inventories`).
- `Host` – Mandatory. When set by downloaded policy (or fail-over locations list), this is normally the host name or fully-qualified domain name of the inventory beacon. If you are setting this value manually, you may also use an IPv4 or IPv6 (unique global or unique local) address.
- `Port` – Mandatory. As this has no default value, you must specify this setting to suit your environment (typically port 80 for HTTP and port 443 for HTTPS).

- `User` – If omitted (or left with a blank value), anonymous authentication is used for uploads to this reporting location.
- `Password` – Stores an encrypted copy of the password needed when Windows authentication is specified for the upload.

Tip: *Since this value is encrypted, it cannot be used when manually editing an* `ndtrack.ini` *configuration file. Use of this setting in a manually-edited file dictates anonymous authentication.*

- `Priority`
- `AutoPriority`.

For an alternative more suited to command lines, see `UploadLocation`.

Values

Values / range	Requires a subkey that uniquely identifies the reporting location on an individual inventory beacon.
Default value	None.

Example values	`[Registry]\ManageSoft\Common\`
	`UploadSettings\{8909c9ba-8492-420e-b6e0-100ecf115b0a}`
	`Protocol=http`
	`Name=Server Room Reporting Location`
	`Directory=ManageSoftRL`
	`Host=WIN-KCPMHPHA0GR`
	`Port=80`
	`User=`
	`Password=`
	`Priority=100`
	`AutoPriority=True`

Registry

Installed by	Download of failover settings, or manual configuration
Computer preference	`[Registry]\ManageSoft\Common\UploadSettings\<reporting_location>`

UploadType

Command line | Registry

`UploadType` determines whether the upload agent uploads machine generated files or user generated files.

Values

Values / range	`Machine`, `User`

Default value	If running as the `SYSTEM` user:
	`Machine`
	If runner as any other user:
	`User`

Example values	`Machine`

Command line

Tool	Upload component (`ndupload`)
Example	`-o UploadType=Machine`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion`

UploadUser

Command line | Registry

`UploadUser` provides the username of the account required to access the location to which files are to be uploaded.

It is rare to specify this preference. In general, the credentials for bootstrapping are embedded in the upload URLs; and for normal operation, the inventory beacon downloads a failover list of all available inventory beacons (which normally all have IIS configured for anonymous authentication). The FlexNet inventory agent saves the contents of this list into the registry on the managed device, under the `DownloadSettings` and `UploadSettings` keys. There may be several keys thereunder, one for each download (or upload) location included in the failover list.

Therefore when (as is normal) `UploadUser` is not defined, the upload agent attempts to use credentials saved in the `[Registry]\ManageSoft\Beacon\UploadSettings` group of keys.

Values

Values / range	Valid user name

Default value	(No default.)
Example values	`JoSmith`

Command line

Tool	Upload component (`ndupload`)
Example	`-o UploadUser=JoSmith`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Uploader\CurrentVersion`

UsageDirectory

Registry

`UsageDirectory` specifies the directory under which a local cache is created for application usage data before it is uploaded from the inventory device, most often to a convenient inventory beacon.

Important: *This preference is set during installation of the FlexNet inventory agent, and should not be changed.*

Values

Values / range	Valid location, as set during installation

Default value	Windows devices:
	`$(CommonAppDataFolder)\ManageSoft Corp\ManageSoft\Usage Agent\UsageData`
	UNIX-like devices:
	`$(CommonAppDataFolder)/managesoft/usageagent/usagedata`

Registry

Installed by	Installation of the FlexNet inventory agent.

Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

UseAddRemove

Command line | Registry

This preference applies only for inventory devices running a Windows operating system.

`UseAddRemove` determines whether the application usage component makes use of the Add/Remove Programs store for identifying executables to monitor:

- When set to `True`, the application usage component uses the Add/Remove Programs data to try to identify application components to monitor
- When set to `False`, the application usage component ignores Add/Remove Programs. This is the default setting, because many publishers' entries here are not reliable – for example, settings may include incomplete installation paths, particularly where the common part of the path is shared by many products from the same publisher.

Tip: *This preference may be changed by downloaded device policy where this inventory device is included in a defined target for which one of the following options has been selected:*

- ***Allow application usage tracking on these targets*** *sets this preference to* `True`
- ***Do not allow application usage tracking on these targets*** *sets this preference to* `False`.

For more information, see FlexNet Manager Suite Help > Discovery and Inventory > Creating a Target.

Values

Values / range	Boolean (true or false)
Default value	`False`
Example values	`True`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o UseAddRemove=True`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

UseManualMapper

Command line | Registry

`UseManualMapper` specifies whether the application usage component makes use of special settings manually recorded in the registry of the inventory device (these settings are collectively known as the "manual mapper"):

- When set to `True`, the application usage component uses data from the manual mapper registry keys to identify executables to monitor and report
- When set to `False`, the application usage component ignores the additional registry keys.

Values

Values / range	Boolean (true or false)
Default value	`False`
Example values	`True`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o UseManualMapper=True`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

UseMSI

Command line | Registry

`UseMSI` specifies whether the application usage component examines the platform-specific native package repositories for executables to monitor:

- When set to `True`, the application usage component reads application data found in the native package repositories (MSI, RPM, or PKG formats)
- When set to `False`, the application usage component ignores the native package repositories.

Values

Values / range	Boolean (true or false)
Default value	`True`
Example values	`False`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o UseMSI=False`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

User

Registry

`User` stores the account name required for authentication during transfer of files between the managed device and the inventory beacon.

When configured by the failover list generated by an inventory beacon, the placeholders *<reporting_location>* and *<distribution_location>* take the form of GUIDs that identify the relevant location on the particular inventory beacon (for example, `{8909c9ba-8492-420e-b6e0-100ecf115b0a}`). In contrast, if you are manually configuring `UploadSettings` in an `ndtrack.ini` file for the lightweight inventory scanner on UNIX-like platforms, you may use any string of ASCII characters (excluding white space) that is unique within the context of the `ndtrack.ini` file for these locations.

Values

Values / range	Valid user name.
Default value	For FTP protocol only: `Anonymous` Otherwise, there is no default.
Example value	`NewUser`

Registry

Installed by	Failover list for inventory beacons, or manual configuration

Computer preference	For uploads:
	`[Registry]\ManageSoft\Common\UploadSettings\<reporting_location>`
	For downloads:
	`[Registry]\ManageSoft\Common\DownloadSettings\<distribution_location>`

UserDefinedOracleHome

Command line | Registry

`UserDefinedOracleHome` is available only on UNIX-like platforms, and is only relevant when a symbolic link was included in the start-up path for an Oracle database instance targeted for inventory collection by the locally-installed tracker (`ndtrack`). The use of a symbolic link can 'mask' the database instance so it is not visible to the tracker, and inventory cannot be collected. There are two ways you can work around this:

- You can ensure that the Oracle home specified in the `/etc/oratab` file represents the `ORACLE_HOME` path used to start the database instance. With this work-around, no other settings are needed, and `UserDefinedOracleHome` may be set to `false` if you so desire.
- The account running the database instance (say *`OSUser4Oracle`*) may set an environment variable within its login profile specifying the `ORACLE_HOME` path (including the symbolic link) which was used to start the database instance. To test this setting, the following command should display the correct `ORACLE_HOME` path:

  ```
  su -OSUser4Oracle -c "echo \$ORACLE_HOME"
  ```

  Tip: *If this environment variable is set for any account on the database server, it is applied to all database instances started by the same account on this server. Any mismatch between the (non-empty) environment*

variable, and the actual path used to start any of these database instances, prevents the collection of database inventory from the mismatched instance by the locally-installed inventory component (`ndtrack`). Conversely, you can prevent the environment variable option being used for all accounts on the target Oracle server by setting the `UserDefinedOracleHome` preference (details of this preference are included in Gathering FlexNet Inventory.

When `UserDefinedOracleHome=true` (or when the setting is omitted, with default `true`), the tracker (in addition to attempting its other normal detection methods) attempts to recover the value of the `$ORACLE_HOME` environment variable for the account running the database instance. If this attempt succeeds, the value recovered replaces any value for `ORACLE_HOME` for this instance collected by any other means (for details of the methods used to detect the `ORACLE_HOME` value, see *Oracle Discovery and Inventory* in *FlexNet Manager Suite System Reference*).

If for some reason you wish to prevent the tracker checking for this environment variable, set `UserDefinedOracleHome=false` on the target device. However, be aware that if the value of `ORACLE_HOME` cannot be determined for a database instance, Oracle inventory cannot be collected for the database instance by the locally-installed tracker.

***Important:** This preference controls behavior of the tracker across all Oracle database instances running on the current server (inventory device). If it happens that you have used multiple accounts for starting separate database instances on this server, and `UserDefinedOracleHome=true`, the tracker searches for the `$ORACLE_HOME` environment variable for each of these accounts, and for all of the database instances started by each of them. Since the priority order of data sources for the Oracle home path for each database instances is:*

1. *`$ORACLE_HOME` environment variable in the account starting and running a database instance on this server*
2. *The `/etc/oratab` file value for the `ORACLE_HOME` path*
3. *The absolute path in use by the process currently running the database instance,*

this means that any mismatch between the `$ORACLE_HOME` environment variable and the path actually used to start and run the database instance causes database inventory collection to fail. This includes (for example) having an environment variable that identifies a symbolic link used for one database instance, even after a possibly-different database instance has been re-started by the same account but using an absolute path. A complete match (with either a symbolic link in both places, or an absolute path in both places) is required for every database instance.

Values

Values / range	Boolean (`true` or `false`, case insensitive)
Default value	`true` This is the default behavior when the preference is omitted.
Example values	`False`

Command line

Tool	Tracker component (`ndtrack`)

Example	`-o UserDefinedOracleHome=false`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

UserHardware

Command line | Registry

`UserHardware` allows you to track hardware (in user-based inventories), either using Windows Management Instrumentation (WMI) or native APIs. If WMI is available, it is used for tracking.

This preference is only effective when running in the user context. To track hardware in the machine context, use `Hardware`.

When set to `True`, allows the tracking of hardware inventory. When set to `False`, does not track hardware inventory.

Values

Values / range	Boolean (`True` or `False`).
Default value	`False`
Example values	`True`

Command line

Tool	Inventory agent (`ndtrack`)
Example	`-o UserHardware=True`

Registry

Installed by	Code internals, or manual configuration
Computer preference	N/A—use `Hardware`

UserInteractionLevel (installation component)

Command line | Registry

`UserInteractionLevel` defines the degree of user interaction with the installation component (this is controlled separately from the inventory component). The preference setting for this component also controls behavior during device reboot. Possible values are:

- `Full`: Installation activities operate in full interactive mode. The user has full control over a package's installation options, and will see all dialogs during the download, installation and uninstall phases. This is generally not encouraged for packages used with the FlexNet inventory agent.
- `Auto`: Installation activities are fully displayed, but no user interaction is required unless an error occurs. Installation proceeds automatically, using the default install values. Again, this is not generally recommended for packages used with the FlexNet inventory agent.
- `Quiet`: No user interface is displayed during operation, and no user feedback or interaction is available.
- `Status`: Only status dialogs are displayed (for example, progress dialogs).

Note: *A synonym (or alias) for this preference is* `UILevel`.

Values

Values / range	`Full`, `Auto`, `Quiet`, `Status`
Default value	`Full` (This default was appropriate to previous scenarios for the installation component, and should probably be changed now.)
Example values	`Quiet`

Command line

Tool	Installation component (`ndlaunch`)
Example	`-o UserInteractionLevel=Quiet`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)

Computer preference	In order of precedence: • `[Registry]\ManageSoft\Launcher\CurrentVersion` • `[Registry]\ManageSoft\Common`

UserInteractionLevel (inventory agent)

Command line | Registry

`UserInteractionLevel` defines the user interaction method of the `ndtrack` agent. Possible values are:

- `Full`: The `ndtrack` agent operates in full interactive mode.
- `Auto`: When `ShowIcon` is `True`, the `ndtrack` agent icon displays during inventory activities. The user is able to double-click the icon to access the `ndtrack` agent user interface. When `ShowIcon` is `False`, a progress bar displays during inventory activities.
- `Quiet`: The `ndtrack` agent is not displayed during operations, and no user feedback or interaction is available.
- `Status`: Only status dialogs are displayed (for example, progress dialogs).

Values

Values / range	`Full`, `Auto`, `Quiet`, `Status`
Default value	`Status`
Example values	`Quiet`

Command line

Tool	Inventory agent (`ndtrack`)
Example	`-o UserInteractionLevel=Quiet`

Registry

Installed by	Manual configuration

Computer preference	In order of precedence: • `[Registry]\ManageSoft\Tracker\CurrentVersion` • `[Registry]\ManageSoft\Common`

UserInventoryDirectory

Command line | Registry

`UserInventoryDirectory` defines the location for the user inventories on the computer device.

Note: *The FlexNet inventory agent uses this option only for user-based inventory when it is executing on a computer device in local mode. Local mode is set automatically when the base directory for the executable matches the value stored in the registry key HKLM\Software\ManageSoft Corp\ManageSoft\EtcpInstallDir. (This means that this folder is not used for zero-footprint inventory, whether collected by the FlexNet inventory agent or the light-weight FlexNet Inventory Scanner. For zero footprint inventory collection, see UserZeroTouchDirectory.)*

Values

Values / range	Valid location.
Default value	`$(AppDataFolder)\ManageSoft Corp\ManageSoft\Tracker\Inventories`
Example values	`C:\temp`

Command line

Tool	Inventory agent (`ndtrack`)
Example	`-o UserInventoryDirectory=C:\ManageSoft Corp\ManageSoft\Tracker\Inventories`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

UserProcessesOnly

Command line | Registry

`UserProcessesOnly` specifies whether the application usage component should exclude applications run by `SYSTEM`/`root` in its tracking:

- When set to `True`, the application usage component tracks applications run by users *other than* `SYSTEM` (on Windows devices) or `root` (on UNIX-like devices)
- When set to `False`, the application usage component tracks applications run by all accounts, including `SYSTEM` or `root` (on appropriate platforms).

Values

Values / range	Boolean (true or false)
Default value	`True`
Example values	`False`

Command line

Tool	Application usage component (`mgsusageag`)
Example	`-o UserProcessesOnly=False`

Registry

Installed by	Installation of FlexNet inventory agent, or manual configuration
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion`

UserScheduleDirectory

Command line | Registry

`UserScheduleDirectory` applies only on Windows devices.

Determines the folder where the user schedules are stored. A user schedule is run for the specified user account on the machine where the schedule resides.

Note: *Changing this preference is not recommended.*

Values

Values / range	Valid folder and path.
Default value	`$(CommonAppDataFolder)\ManageSoft Corp\ ManageSoft\Schedule Agent\Schedules`
Example values	`C:\Program Files\Flexera Software\schedule agent\ UserSchedules`

Command line

Tool	Scheduling component (`ndschedag`)
Example	`-o UserScheduleDirectory="C:\Program Files\Flexera Software\schedule agent\ UserSchedules"`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)
Computer preference	`[Registry]\ManageSoft\Schedule Agent\CurrentVersion`

UserZeroTouchDirectory

Command line | Registry

`UserZeroTouchDirectory` specifies the directory where user inventory files are written (temporarily, pending upload) during a remote ("zero touch") inventory gathering process. If the upload (called as part of the inventory scanning process) proceeds normally, each temporary file is cleaned up after upload.

Note: *The FlexNet inventory agent references this setting for any user-based inventory involving remote execution (zero touch inventory gathering). Remote mode is set automatically when the registry key `HKLM\Software\ ManageSoft Corp\ManageSoft\EtcpInstallDir` does not exist or does not match the base directory for the executable. The registry key is typically missing during zero footprint inventory collection, because the FlexNet inventory agent has not been permanently installed on the managed device. (This means that, when the FlexNet inventory agent is locally installed one the managed device, this folder is not used. Instead, in this case see UserInventoryDirectory.) Also note that the default for the FlexNet inventory agent is different than the default for the*

lightweight FlexNet Inventory Scanner.

Values

Values / range	Any valid folder
Default value	`%temp%\FlexeraSoftware\` This is the temporary directory for the account running the inventory scan. (For the installed FlexNet inventory agent, this is the same default value as for `MachineZeroTouchDirectory`.)
Example values	`C:\temp`

Command line

Tool	Inventory agent (`ndtrack`)
Example	`-o UserZeroTouchDirectory=C:\temp`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

Version

Registry

`Version` identifies the version of an application in the manual mapper for the application usage component. After usage is detected and uploaded, this version appears in the **Raw Software Usage** page in the web interface for FlexNet Manager Suite. This registry key must be created manually, within a node for the chosen application that has also been inserted manually (and shown below as *`Application node`*).

Tip: *If this value for `Version`, together with the value for `Application`, are exact matches for the application version and name recorded in installer evidence for the application, the set-up can be simplified a little, because this mapping of the executable to the application automatically matches through the known installer evidence. (If not, this manual mapper entry can also be manually linked to the application record.)*

Values

Values / range	Valid text character string, which may include spaces as required.
Default value	No default value.
Example values	`1.0`

Registry

Installed by	Manual configuration only.
Computer preference	`[Registry]\ManageSoft\Usage Agent\CurrentVersion\Manual Mapper\`*`Application node`*

VersionInfo

Command line | Registry

When `VersionInfo` is set to `True`, the tracker (`ndtrack` executable) includes file version header information in the inventory collected on Microsoft Windows systems.

When set to `False`, the tracker does not include file version header information in the inventory.

This preference is ignored for UNIX-like systems.

Values

Values / range	Boolean (`True` or `False`).
Default value	`True`
Example values	`False`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o VersionInfo=False`

Registry

Installed by	Code internals, or manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

WinDirectory

Command line | Registry

`WinDirectory` references the path to the Windows folder. You can also use `WindowsFolder`, which points to the same folder.

Warning: *Changing either of these values may have unpredictable results. The two preferences are initialized at start-up, and are set from the Windows environment. If you manually change either one, the change is not synchronized, and the behavioral impacts are unpredictable.*

This preference is ignored for UNIX-like platforms.

Values

Values / range	
Default value	`C:\Windows`
Example values	`C:\Windows`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o WinDirectory=C:\Winnt`

Registry

Installed by	Predefined within FlexNet inventory agent / Microsoft Windows
Reference as:	`$(WinDirectory)`

WMI

Command line | Registry

When `WMI` is set to `True`, the Windows Management Instrumentation (WMI) tracking is specified as the preferred option for tracking hardware. In this case, if WMI is not available (and the `Hardware` preference is set to `True`), the tracker (`ndtrack` executable) attempts to track hardware using a native API.

When set to `False`, the tracker uses another tracking mechanism instead of WMI.

This preference is ignored for UNIX-like platforms.

Values

Values / range	Boolean (`True` or `False`).
Default value	When taking machine-based inventory, the default behavior is: `True`
Example values	`False`

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o WMI=False`

Registry

Installed by	Manual configuration
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

WMIConfigFile

Command line | Registry

`WMIConfigFile` defines the location of the Windows Management Instrumentation (WMI) configuration file, used to inform the tracker (`ndtrack` executable) what hardware components it should track. This is only used if `WMI` is `True`.

This preference is ignored for UNIX-like platforms.

Values

Values / range	Valid location.
Default value	`$(ProgramPath)\wmitrack.ini`
Example values	The value should point to a valid `.ini` file that contains WMI configuration.

Command line

Tool	Inventory component (`ndtrack`)
Example	`-o WMIConfigFile=C:\Program Files\ManageSoft\Tracker\wmitrack.ini`

Registry

Installed by	Installation of FlexNet inventory agent (computer preference)
Computer preference	`[Registry]\ManageSoft\Tracker\CurrentVersion`

11

File Formats

The content in the following topics is common to all methods of FlexNet inventory collection, except as specifically noted.

Application Usage Files (.mmi)

The application usage component is a long-running service on both Windows and UNIX-like operating systems. It queries the current set of running processes (by default) every minute, and tracks all process creation and termination over a period of time. The process execution is summarized for each user of the inventory device. Initially, the data is held in memory, and is subsequently written to a usage file on the local hard drive. On a specified cycle, by default once every 8 hours, the saved usage files are processed and written into an XML file with the extension `.mmi`. By default, a `gzip` archive is created for each `.mmi` file, so that it becomes a `.mmi.gz` file.

Details for usage (.mmi) files

Server location	On each inventory device where the FlexNet inventory agent is locally installed, and tracking of application usage is enabled.
Folder	The `.mmi` files are saved (and by default archived) in: • On Windows devices: `C:\ProgramData\ManageSoft Corp\ManageSoft\Common\Uploads\UsageData` • On UNIX-like devices: `/var/opt/managesoft/uploads/UsageData`. As soon as the `.mmi` (or `.mmi.gz`) files are prepared, the application usage component invokes the upload component (uploading usage tracking files does not follow the upload schedule set on the inventory device). Files that have been uploaded successfully to an inventory beacon are then removed from these locations. These behaviors mean that the `.mmi[.gz]` files may be quite transient in the above folders.

Updated	The application usage files for upload are generated after each time interval specified in the `UploadPeriod` preference (by default, 8 hours), collecting the usage data that has been archived during the day. A weekly usage summary is also uploaded.
File name format	*deviceName* `at` *timestamp*`.mmi` (with trailing `.gz` when archived) Example: `meldskasherlok at 20200310T172738.mmi`
Format	XML (text).

Sample file

In the following sample, line numbers have been added for reference only; only a single application usage record is included; and some lines have been wrapped for presentation.

```
(1) <?xml version="1.0" encoding="UTF-8"?>
(2) <!DOCTYPE USAGE PUBLIC "USAGE" "http://www.managesoft.com/usagelog.dtd">

(3) <USAGE VERSION="1.00" USAGEAGENT="14.2">
(4)     <MD NAME="meldskasherlok" DOMAIN="example.com"
            GUID="{3165052f-d755-478a-8981-9bc72f20e9bf}">
(5)                 <USAGE USER="asherlok" DOMAIN="example.com"
                     TYPE="WeeklySummary" STARTDATE="20200309T000000"
                     GUID="{43210673-a7e8-494a-a767-4ac2f61c81fb}">
(6)                         <APPUSAGE NAME="Adobe Reader XI (11.0.10)"
                            VERSION="11.0.10" FILENAME="AcroRd32.exe"
                            FILECOMPANY="Adobe Systems Incorporated"
                            FILEDESCRIPTION="Adobe Reader "
                            FILEPRODUCTNAME="Adobe Reader"
                            FILEPRODUCTVERSION="11.0.10.32"
                            FILEVERSION="11.0.10.32"
                            LONG_FILENAME="C:\Program Files (x86)\Adobe\Reader
                                11.0\Reader\AcroRd32.exe"
                            TOTALRUNTIME="298374"
                            TOTALACTIVE="0"
                            SESSIONS="2"
                            DAYS="1"/>
(7)             </USAGE>
(8)     </MD>
(9) </USAGE>
```

The components in the catalog file, line by line, are:

- (1) and (2): The XML header. The FlexNet inventory agent has built-in knowledge of the usage file structure, and does not use the DTD reference.
- (3) and (9): Version information for the file format, and the version of the application usage component that collected the data.
- (4) and (8): Enclosing tags and identifier for the managed device (a historical name for the inventory device) sending the reported usage. The `GUID` attribute is the Active Directory GUID of the computer (inventory device) in the

domain, and may be null if the device is not joined to a domain (such as a UNIX-like platform).

- (5) and (7): The enclosing tags for the set of usage records. This set is unique to the end-user, and covers the period shown. Notice that there may be multiple `USAGE` sets if different users work on this inventory device, and these may include service accounts such as `LOCAL SERVICE`. The `GUID` attribute is the Active Directory GUID of the end-user executing the application, where available.
- (6): A record of application usage. Normally an `.mmi` file contains several of these, all of the same format. See the table below for details of the attributes included in this record.

Attribute	Description	Example
NAME	The application name identified in the installer evidence.	VMware Workstation
VERSION	The application version identified in the installer evidence.	8.0.3.29699
FILENAME	The name of the executable being monitored for usage.	vmware-vmx.exe
FILECOMPANY	The company name appearing in the executable file header.	VMware, Inc.
FILEDESCRIPTION	The description of the software taken from the executable file header.	VMware Workstation VMX
FILEPRODUCTNAME	The product named in the executable file header.	VMware Workstation
FILEPRODUCTVERSION	The product version identified in the executable file header.	8.0.4 build-744019
FILEVERSION	The executable file version identified in its file header.	8.0.4 build-744019
LONG_FILENAME	The full path to the executable file.	C:\Program Files (x86)\VMware\VMware Workstation\x64\vmware-vmx.exe
TOTALRUNTIME	Total time in seconds that the application was running within the sample period. This time may be accumulated over one or more sessions in the period.	482486
SESSIONS	Number of times the application has started (and stopped) within the sample period.	4
TOTALACTIVE	Total time in seconds the application was active in the foreground within the sample period, over one or more sessions.	1

Attribute	Description	Example
DAYS	The number of distinct days within the sample period when the application was used.	3

The uploaded data is imported into the inventory database in the following tables, all of which are detailed in *FlexNet Manager Suite Schema Reference*:

- `ComputerUsage` table
- `SoftwareFileUsage` table
- `SoftwareUsagePerWeek` table.

Catalog files (.osd)

A catalog file is small, usually around 1KB, and thus provides a fast way for the FlexNet inventory agent to check whether downloads of larger files are required (that is, whether the file that the catalog points to has changed). In this description, that larger file is called the "target file" to distinguish it from the catalog file. The catalog and its target file are together called a 'package'.

The catalog file includes:

- A pointer to the target file
- An MD5 checksum for the target file
- The software version
- Optionally, licensing details
- The names and versions of any dependent packages.

The format of the catalog file was derived from the W3C's Open Software Description (OSD) format, described at http://www.w3corg/TR/NOTE-OSD.html. However, the catalog file format ignores some elements of the OSD proposal, and adds others (through a namespace declaration with a related DTD file).

Details for package (.osd) files

Server location	Generated on demand by (and saved on) inventory beacons.

Folder	On inventory beacon: In the appropriate subdirectories under *%CommonAppData%*\Flexera Software\Staging\Common\. Relevant subdirectories for catalogs include: • ClientConfiguration • ClientSettings • InventorySettingsConfiguration • Packages • Schedules. ***Tip:*** *Catalog files exist only after an installed FlexNet inventory agent has requested the relevant download. For example, the Default Machine Schedule.osd catalog is created only after a managed device has requested a schedule update.* On Windows devices after download: On UNIX-like devices after download:
Updated	The catalog files are initially generated (and cached) when requested by a managed device, using the latest inventory beacon policy downloaded from the central application server. Thereafter each catalog is checked on demand for changes to the target file requested by the installed FlexNet inventory agents.
File name format	*packageType*.osd Example: Default Machine Schedule.osd
Format	XML (text).

Sample file

In the following sample (which is a catalog for a schedule), line numbers have been added for reference only; some lines have been elided to shorten them for presentation.

```
(1) <?xml version="1.0" encoding="UTF-8"?>
(2) <!DOCTYPE SOFTPKG SYSTEM "http://www.managesoft.com/softpkg.dtd">
(3) <?XML::NAMESPACE HREF="http://www.managesoft.com/managesoft.dtd" AS="NETDEPLOY"?>
(4) <SOFTPKG NAME="Default Machine Schedule" TYPE="Schedule" VERSION="1,0,0,0"
FORMAT="1.5">
(5)         <TITLE>Default Machine Schedule</TITLE>
(6)         <ABSTRACT>.
(7) </ABSTRACT>
(8)         <NETDEPLOY::PROPERTY NAME="PackageTransferSize">0</NETDEPLOY::PROPERTY>
(9)         <NETDEPLOY::USERDOMAIN DEFAULT="public" USERSELECT="True" INHERIT="True"/>
(10)        <IMPLEMENTATION>
(11)                <NETDEPLOY::DIGEST ALGORITHM="MD5" VALUE="e61e527...3033"/>
(12)                <NETDEPLOY::SYMBOL NAME="SelfHeal" VALUE="True"/>
```

```
(13)                <CODEBASE HREF="Default%20Machine%20Schedule.nds"/>
(14)        </IMPLEMENTATION>
(15) </SOFTPKG>
```

The components in the catalog file, line by line, are:

- (1) and (2): The XML header. The FlexNet inventory agent has built-in knowledge of the catalog file structure, and does not use the DTD reference.
- (3): A namespace definition that allows the use of additional elements through the second DTD file. Once again, knowledge of this structure is built into the FlexNet inventory agent, and the DTD reference is not used.
- (4) and (15): Catalog (`SOFTPKG`) start and end tags, with attributes for the name, type, version, and format.
- (5): A display title for the catalog.
- (6): An abstract that may be used to summarize the purpose of the package.
- and (8), (9) and (15), (16) and (22), and (23) and (27): Event start and end tags, with the most helpful attribute being the `NAME`.
- (6), (10), and similarly in later events: The action to execute when any of the following triggers fires. The relevant component of the FlexNet inventory agent is identified, along with any command line parameters to inject.
- (7), (11)-(14), and similarly in later events: Details of the trigger (or schedule pattern) when the above action should be executed. The parameters include typical scheduling details, such as:
  - The start of the time window within which the trigger should fire, along with the length of the time window in seconds (this time window allows for randomization across different devices, so that they do not all impact the network at the same time)
  - Whether the event should repeat during the day (with the delay in seconds, so that for example the policy update repeats every 12 hours)
  - Optionally, additional triggers to fire the same event when a new user logs on, when the device reconnects to the network, or immediately on machine reboot.

Policy Files (.npl)

Files with `.npl` extensions contain policy information for use by the FlexNet inventory agent (whether deployed in the Adopted case or the Agent third-party deployment case).

Tip: *Policy files are not available to FlexNet inventory core components, in any of the Zero-footprint, FlexNet Inventory Scanner, or Core deployment scenarios.*

Policy files are generated on demand by inventory beacons, based on information downloaded from the central application server. Each file contains links to all the information needed by the FlexNet inventory agent for normal operation. A separate policy file is generated when the first request is received from each installed FlexNet inventory agent, and the newly-generated policy (and all its dependent files) are cached against future requests. If the inventory beacon receives updated instructions from the central application server, the cache is cleared, such that new policies, incorporating the changed information, are generated when fresh requests are received.

You should not edit policy files, but viewing them can help you to determine that the policy information being distributed to managed devices is as you expect. For example, if the policy cache does not contain a policy for a target device that you expect to see, it may be because:

- The managed device is requesting policy from a different inventory beacon
- The managed device is yet to request policy from this inventory beacon since updated data from the central application server caused the policy cache to be cleared
- The managed device is not correctly configured, or is malfunctioning.

Details for policy files created during policy distribution

Server location	Generated on demand by (and saved on) inventory beacons.
Folder	*%CommonAppData%*\Flexera Software\Staging\Common\Policies
Updated	Automatically on first request from a managed device. Calculated policy (and all dependent files) are cached against future requests. The cache is cleared when the inventory beacon receives updated data from the central application server.
File name format	*deviceName-UID*.npl (the unique identifier is derived from the IP addresses applicable to the managed device).
Format	XML (text).

Sample file

In the following sample, line numbers have been added for reference only; and lines have been shortened with placeholders for reproduction.

```
(1) <?xml version="1.0" encoding="utf-8"?>
(2) <!DOCTYPE Policy PUBLIC "Policy" "http://www.managesoft.com/policy.dtd">

(3) <Policy AppliesTo="*" Source="ManageSoft" Version="0">
(4)   <Policy GUID="GUID-for-policy-version" Source="ManageSoft" Version="0">
(5)     <Package href="PathToSchedule" ObjectID="IDForSchedule" Precedence="0" />
(6)     <Package href="PathToConfig" ObjectID="IDForConfig" Precedence="0" />
(7)     <Package href="PathToInvSettings" ObjectID="IDForInvSettings" Precedence="0" />
(8)     <Package href="PathToFailover" ObjectID="IDForFailover" Precedence="0" />
(9)     <Package href="PathToUpgrade" ObjectID="IDForUpgrade" Precedence="0" />
            ...
(10)  </Policy>
(11) </Policy>
```

The policy file contains links to package (.osd) files, and those files identify others that contain the actual content. The FlexNet inventory agent is able to deduce from the package files whether any of the referenced content files have been updated since last downloaded to the managed device by the FlexNet inventory agent. If there are no updates, there are no further downloads, which minimizes network load and maximizes performance. The components in the policy file, line by line, are:

- (1) and (2): The XML header. The FlexNet inventory agent has built-in knowledge of the policy file structure, and does not use the DTD reference.
- (3), (4), (10), and (11): Policy start and end tags, with attributes for version, source, and so on. The attribute `AppliesTo="*"` identifies machine policy (all users), and must not be changed.
- (5): A link to the schedule for inventory collection by the FlexNet inventory agent. The schedule is saved on the inventory beacon in *%CommonAppData%*`\Flexera Software\Staging\Common\Schedules`. (Within the policy file, this path is referenced only as `/Schedules/`, with the file name appended.) This link is to the `.osd` file in the folder, and this in turn links to an `.nds` file that contains details of the schedule itself.
- (6): A link to the client configuration file that contains initial settings for FlexNet inventory agent. Client configurations are saved on the inventory beacon in *%CommonAppData%*`\Flexera Software\Staging\Common\ClientConfiguration`. (Within the policy file, this path is referenced only as `/ClientConfiguration/`, with a subfolder and the file name appended. The subfolders hold distinct sets of client configurations, separated by the appropriate UID.) This link is to the `.osd` file in the folder, and this in turn links to an `.ndc` file that contains details of the client configuration itself. Client configuration is minimal, mainly declaring the starting point for use of the registry on the managed device by the FlexNet inventory agent, any settings for the `IncludeDirectory` preference, and whether usage tracking is enabled.
- (7): A link to the inventory settings file that extends the inventory collection capabilities of the FlexNet inventory agent. Inventory settings packages for download and installation on managed devices are saved on the inventory beacon in `%CommonAppData%\Flexera Software\Staging\Common\InventorySettingsConfiguration\InventorySettings`. (Within the policy file, this path is referenced only as `/InventorySettingsConfiguration/InventorySettings/`, with the `.osd` file name appended.) The `.osd` package file links to an `.ndc` file, which in turn points to an `InventorySettings.xml` file that contains extensions for inventory gathering, including (for example) the queries used for Oracle Database inventory gathering.

  Tip: *In operation for the Zero-footprint case, the inventory beacon uses an installed copy saved in* *%ProgramFiles%*`\Flexera Software\Inventory Beacon\RemoteExecution\Public\Inventory`.
- (8): A link to the failover settings, saved in *%CommonAppData%*`\Flexera Software\Staging\Common\ClientSettings\Default Failover Settings` (again, everything down to the `Common` folder is assumed in the policy file). As is common in the policy file, the link is to an `.osd` package file that links to an `.ndc` file containing the data. In this case, the data consists of a list of registry settings where the FlexNet inventory agent can save connection details for every known inventory beacon. If its preferred inventory beacon is unavailable, the FlexNet inventory agent can attempt inventory uploads to another inventory beacon that responds to an initial contact (inventory beacons saved in the failover list are configured for anonymous authentication). Notice that the failover settings are used independently by the `ndupload`, `ndtrack`, and `ndlaunch` components of the FlexNet inventory agent. (It is the `ndlaunch` component that downloads policy, and downloads and checks all `.osd` packages.)
- (9): The first of several links to upgrade packages for the FlexNet inventory agent on supported platforms, with the first link being for Microsoft Windows, and following packages for named platforms. These are stored on the inventory beacon under *%CommonAppData%*`\Flexera Software\Staging\Common\Packages\Flexera\Upgrade\` (with the path down to the `Common` folder being assumed in the policy file). Folders for approved version(s) of the FlexNet inventory agent, and for the revision to packages for that version, are included in the path; along with a platform-specific folder. The link concludes with the `.osd` file name, and this in turn points to the

elements needed by for self-updating of the FlexNet inventory agent. This line of the policy file is repeated once for each supported platform on which the FlexNet inventory agent runs. The installed FlexNet inventory agent has the intelligence to download only its own required version, and only when an upgrade is required.

Schedule Files (.nds)

Files with the `.nds` extension contain data about schedules for use by the FlexNet inventory agent (in both the Adopted case and the Agent third-party deployment case).

A single schedule file is generated by inventory beacons each time changed schedule information is downloaded from the central application server. The file is cached against future requests, and a package file (`.osd`) is created at each update that references the new schedule. (The package file is referenced in the policy file downloaded by each FlexNet inventory agent, for which see Policy Files (.npl).) A single schedule file is used in common by all FlexNet inventory agents.

You should not edit schedule files, but viewing them can help you to determine that the schedule information distributed to managed devices is as you expect.

Details for schedule (.nds) files created during schedule changes

Server location	Generated on demand by (and saved on) inventory beacons.
Folder	On inventory beacon: *%CommonAppData%*`\Flexera Software\Staging\Common\Schedules` On Windows devices after download: `%CommonAppData% \ManageSoft Corp\ManageSoft\Schedule Agent\Schedules` On UNIX-like devices after download: `/var/opt/managesoft/scheduler/schedules`
Updated	Automatically each time that inventory beacon policy downloaded from the central application server includes a change to the schedule for installed FlexNet inventory agents.
File name format	`Default Machine Schedule.npl`
Format	XML (text).

Sample file

In the following sample, line numbers have been added for reference only; some lines have been wrapped for reproduction; and others have been elided when they repeat similar content.

```
(1) <?xml version="1.0" encoding="utf-8"?>
(2) <!DOCTYPE Schedule PUBLIC "ManageSoft Schedule"
         "http://www.managesoft.com/schedule.dtd">
(3)
(4) <Schedule SCHEDSCHEMA="60" NAME="Default Machine Schedule"
```

```
          TIMEDATESTAMP="20160204T151801Z">
(5)        <Event NETWORK="false" NAME="Generate Inventory"
            ID="{de9cddcd-53ac-4796-b02c-f00764f00a1f}"
            CATCHUP="Never" IDLEDURATION="60">
(6)                <LogicalCommand PARAM=" -o UserInteractionLevel=Quiet"
                   COMPONENT="Tracker" ACTION="Report" />
(7)                <Trigger TIMESTART="080030" TYPE="Daily" TYPE_PARAM="1"
                   TIMEWINDOW="28800" DATESTART="20100101" />
(8)        </Event>
(9)        <Event NETWORK="false" NAME="Update Machine Policy"
            ID="{30c8899c-9eab-4f07-9160-d8610b27f67f}"
            CATCHUP="Never" IDLEDURATION="0">
(10)             <LogicalCommand PARAM="-t Machine -o UserInteractionLevel=Quiet"
                   COMPONENT="PolicyClient" ACTION="Apply" />
(11)             <Trigger TIMESTART="000500" REPEAT="43200" DURATION="86400"
                   TYPE="Daily" TYPE_PARAM="1" TIMEWINDOW="3600" DATESTART="20160204"
/>
(12)             <Trigger TYPE="Logon" ... />
(13)             <Trigger TYPE="OnConnect" ... />
(14)             <Trigger TYPE="Now" ... />
(15)       </Event>
(16)       <Event NETWORK="false" NAME="Update Client Settings" ... >
(17-21)       ...
(22)       </Event>
(23)       <Event NETWORK="false" NAME="Upload Client Files" ... >
(24-26)     ...
(27)       </Event>
(28) </Schedule>
```

The components in the schedule file, line by line, are:

- (1) and (2): The XML header. The FlexNet inventory agent has built-in knowledge of the schedule file structure, and does not use the DTD reference.
- (4) and (28): Schedule start and end tags, with attributes for the time and date of last change, and schema version (the schema is internal to the `ndschedag` component).
- (5) and (8), (9) and (15), (16) and (22), and (23) and (27): Event start and end tags, with the most helpful attribute being the `NAME`.
- (6), (10), and similarly in later events: The action to execute when any of the following triggers fires. The relevant component of the FlexNet inventory agent is identified, along with any command line parameters to inject.
- (7), (11)-(14), and similarly in later events: Details of the trigger (or schedule pattern) when the above action should be executed. The parameters include typical scheduling details, such as:
  - The start of the time window within which the trigger should fire, along with the length of the time window in seconds (this time window allows for randomization across different devices, so that they do not all impact the network at the same time)
  - Whether the event should repeat during the day (with the delay in seconds, so that for example the policy update

repeats every 12 hours)

- Optionally, additional triggers to fire the same event when a new user logs on, when the device reconnects to the network, or immediately on machine reboot.

WMI Configuration File (wmitrack.ini)

The WMI (Windows Management Instrumentation) configuration file is used on Microsoft Windows platforms to inform the inventory tool (`ndtrack.exe`) what hardware, software and operating system components it should track. This file is referenced by default, but for the installed FlexNet inventory agent, its use may be turned off by setting the `WMI` preference to false (see WMI). For the FlexNet Inventory Scanner case, it is always enabled (since the lightweight FlexNet Inventory Scanner does not check registry entries).

The components to be tracked can be any valid Win32 classes. A full list of classes is provided at https://msdn.microsoft.com/en-us/library/aa394583%28v=vs.85%29.aspx.

You can edit this file to change the items being tracked.

Details of file availability

Machine location	On managed devices (machines where the full FlexNet inventory agent is installed); with a copy saved on inventory beacons.
Folder	On managed devices: `$(ProgramPath)\wmitrack.ini` On inventory beacons: `%ProgramFiles%\Flexera Software\Inventory Beacon\Tracker`
Updated	Installed on managed devices when the FlexNet inventory agent is installed (for example, in the Adopted case). Installed on inventory beacons when the FlexNet Beacon software is installed. This file is never updated automatically.
File name	`wmitrack.ini`
Format	`.ini` file (text, with section headings enclosed in square brackets following by the member settings in plain text).

Sample file

This sample shows all the default settings. Notice that some lines are commented out with the leading semi-colon character.

```
[Win32_ComputerSystem]
Manufacturer
Model
Domain
DomainRole
NumberOfProcessors
NumberOfLogicalProcessors
```

```
TotalPhysicalMemory
Status
UserName

[Win32_ComputerSystemProduct]
IdentifyingNumber
Name
UUID
Vendor
Version

[Win32_OperatingSystem]
Name
Manufacturer
Version
ServicePackMajorVersion
ServicePackMinorVersion
SerialNumber
InstallDate
LastBootUpTime
OSLanguage
FreePhysicalMemory
FreeVirtualMemory
CountryCode
WindowsDirectory
SystemDirectory
Caption
CSDVersion
Status
CSName
OSType
OSArchitecture

[Win32_BIOS]
Manufacturer
Version
ReleaseDate
SerialNumber
BiosCharacteristics
Status

[Win32_Processor]
Description
Manufacturer
Version
ProcessorId
CurrentClockSpeed
CurrentVoltage
```

```
L2CacheSize
Status
MaxClockSpeed
Name
ProcessorType
NumberOfLogicalProcessors
NumberOfCores
DeviceID

[Win32_DiskDrive]
Description
Manufacturer
Model
Size
InterfaceType
Partitions
Status

[Win32_LogicalDisk]
Description
VolumeName
FileSystem
FreeSpace
Size
VolumeSerialNumber
DriveType
MediaType
Status
ProviderName

[Win32_CDROMDrive]
Description
Manufacturer
Drive
Status
Capabilities

[Win32_NetworkAdapter]
Manufacturer
MACAddress
MaxSpeed
Speed
Status

[Win32_NetworkAdapterConfiguration]
Caption
Description
Index
```

```
MACAddress
IPEnabled
DHCPEnabled
IPAddress
DHCPServer
DNSHostName
DNSDomain
DNSServerSearchOrder
DefaultIPGateway
IPSubnet

[Win32_PhysicalMemory]
Capacity
MemoryType
PositionInRow
Speed
Status

[Win32_SoundDevice]
Name
Manufacturer

[Win32_VideoController]
Name
VideoProcessor
DriverVersion
DriverDate
InstalledDisplayDrivers
AdapterRAM

[Win32_VideoConfiguration]
AdapterRAM
AdapterType
Description
HorizontalResolution
MonitorManufacturer
MonitorType
Name
VerticalResolution

[Win32_SystemEnclosure]

;[Win32_USBDevice]
;Caption
;ClassGuid
;Description
;DeviceID
;Manufacturer
```

```
;Name
;Status
;SystemName

[SoftwareLicensingProduct]
ApplicationID
Description
EvaluationEndDate
GracePeriodRemaining
LicenseStatus
MachineURL
Name
OfflineInstallationId
PartialProductKey
ProcessorURL
ProductKeyID
ProductKeyURL
UseLicenseURL

[SoftwareLicensingService]
ClientMachineID
IsKeyManagementServiceMachine
KeyManagementServiceCurrentCount
KeyManagementServiceMachine
KeyManagementServiceProductKeyID
PolicyCacheRefreshRequired
RequiredClientCount
Version
VLActivationInterval
VLRenewalInterval
```

Two FlexNet Kubernetes Agents

The two Kubernetes agents from Flexera are standalone applications specifically designed to capture inventory data from Kubernetes clusters. There are two implementations to choose from:

- The Flexera Kubernetes inventory agent (sometimes called the 'full' Kubernetes agent) is the primary implementation that is recommended for most organizations.
- The lightweight Kubernetes agent is intended for high security environments, omitting some features, automation, and capabilities present in the other agent in order to have the smallest possible footprint, and to provide the maximum manual control of its configuration and operation (this agent is the main focus of this section of the document you are reading).

One instance of either agent is deployed into each Kubernetes cluster as a native containerized application, and is managed using standard Kubernetes tooling. Either agent observes the cluster into which it is deployed, produces inventory files containing the observed data, and uploads the files to an existing FlexNet inventory beacon. The agent collects its information by connecting to the Kubernetes API, and using the `watch` interfaces to subscribe to event streams for the resources it needs to monitor. It extracts the data it needs from the API data and stores it in a local cache (either in persistent storage or in memory), periodically flushing the data out into an inventory (`.ndi`) file. Because both of these agents target the standard Kubernetes API, they can operate with minimal configuration on any platform based Kubernetes version 1.16 or later.

Either agent collects the following information from the cluster where its container is installed:

- Basic cluster metadata:
  - Kubernetes version
  - A unique ID for the cluster
- The nodes that compose the cluster:
  - Hardware resources
  - Serial number of the underlying server (Flexera Kubernetes inventory agent only, optional)
  - Cloud instance metadata for the underlying server (Flexera Kubernetes inventory agent only, optional)
- The Pods that are deployed in the cluster:
  - The images on which the containers are based

- Resource limits applied to containers
- Usage: when, how many, and for how long Pods are used
- Software-identifying annotations applied to Pods
- Kubernetes resources that own Pods for contextualization (optional)

- Data from the IBM License Service about IBM software (in particular, IBM Cloud Paks) running in the cluster (optional)
- Additional software content of images (Flexera Kubernetes inventory agent only, optional).

Supported architectures

The following hardware architectures are supported:

- The x86_64 architecture, which is designed for AMD and Intel 64-bit computers.
- The s390x architecture, also known as "System z", "zSystems", or "z/Architecture", which is a mainframe architecture developed and supported by IBM.
- The ARM64 architecture, also known as the AArch64 architecture, which can run on the Amazon Graviton processors.

Relationship with the FlexNet inventory agent

The two Kubernetes agents are entirely independent of the standard FlexNet inventory agent that collects full hardware and software inventory from a variety of environments (nor do they in any sense *replace* that standard inventory tool). They also have a separate purpose: for example, they do *not* collect inventory of software that may be installed on the servers acting as nodes in the cluster. In general, it is best practice that worker nodes in a Kubernetes cluster run only Kubernetes components; but where software inventory of the node servers is required, the FlexNet inventory agent can be separately installed and operated on those servers.

Restriction: *When either Kubernetes agent is collecting Kubernetes-related inventory for a cluster, and when the FlexNet inventory agent is also deployed to any node server(s), the container inventory feature of the FlexNet inventory agent should not be enabled. That feature targets a locally-running Docket daemon, so that, for nodes running Docker as a container runtime, in this scenario there will be duplicate inventory reports from the two kinds of inventory agents. Because Kubernetes supports a variety of container runtimes, it adds abstractions around the runtime which may prevent the two inventory reports being 'socialized' – that is, recognized as two reports of the one item that should be merged. The result may be a double-counting of containers – to prevent which, do not enable the container inventory feature of the FlexNet inventory agent in this scenario.*

Choosing between the agents

The full Flexera Kubernetes inventory agent is recommended for most environments, because it offers relative ease of use and complete software inventory of the contents of containers. The lightweight Kubernetes agent can be considered for cases where some third-party tool is used to take inventory of container images, and where the additional requirements of the Flexera Kubernetes inventory agent are not a good fit with your enterprise policies governing your Kubernetes environment. The decision between these two agents normally involves a conversation between your ITAM team and the platform team managing your Kubernetes environment.

When choosing between the Flexera Kubernetes inventory agent and the lightweight Kubernetes agent, the following factors may assist:

Factor	Flexera Kubernetes inventory agent	lightweight Kubernetes agent
Configuration	May be automated.	Requires manual specification, either with command-line flags on the installer, or editing of `.yaml` files.
Persistent storage	Requires persistent storage within its container.	Primarily intended to operate without persistent storage (although this can be configured as usual within Kubernetes).
Permissions	Controlled by role, includes read/write options. Requires limited Kubernetes write permissions as part of automated management. Normally run as a root user (especially if calling the FlexNet inventory agent).	Fewest possible permissions, and read-only. Can run as a non-root user.
Operator pattern	Required option (which requires write permissions in the controller).	Not supported.
Container image	Default image supplied, including pre-configuration. Includes standard (Linux) OS layer.	"From scratch" (single, stand-alone binary executable, written in Go), with manual configuration required. Container can be immutable at run time.
Integration with IBM License Service	Supported (off by default, must be enabled).	Supported (off by default, must be enabled).
Additional software inventory	Can inject the FlexNet inventory agent temporarily into a container to collect inventory of ancillary software. Minimizes this process by assessing only one container per image.	Not supported. Other than reporting IBM Cloud Paks (through integration with IBM License Service), third-party tools are required to take inventory of software within containers.
Prerequisite software	The following software tools must be installed: • yq • kustomize • kubectl	The following software tools must be installed: • yq • kubectl

Installation instructions

To download and install the Flexera Kubernetes inventory agent, see the topic "Download Flexera Kubernetes Inventory Agent" in the *Online Help*.

To download and install the lightweight Kubernetes agent, see The Lightweight Kubernetes Agent.

1

The Lightweight Kubernetes Agent

The topics in this chapter provide considerable detail about the lightweight Kubernetes agent:

- Details of its operation (see How the Lightweight Kubernetes Agent Works)
- How to obtain, and install (and if need be, uninstall) the lightweight Kubernetes agent (start from Downloading the Lightweight Kubernetes Agent, and see also following topics)
- All the options that can be configured for it (see Options for the Lightweight Kubernetes Agent).

This is followed by a separate chapter giving details of the inventory files that may be uploaded by either the lightweight Kubernetes agent or the Flexera Kubernetes inventory agent, since both these agents return the same inventory results.

How the Lightweight Kubernetes Agent Works

After you have downloaded and installed the lightweight Kubernetes agent (see Downloading the Lightweight Kubernetes Agent and associated sub-topics), Kubernetes instantiates your container, and the lightweight Kubernetes agent immediately connects to the Kubernetes API, and uses the `watch` interfaces to subscribe to the events streams for the `Node`, `Namespace`, and `Pod` resources in the cluster that it needs to monitor.

More specifically, the lightweight Kubernetes agent (using read-only permissions):

- Reads the `kube-system` `Namespace`, obtaining its UID to use as a cluster identifier (`get namespace`)
- Reads the Kubernetes version
- Watches the `Nodes` resource (`watch nodes`) to extract the hardware resources of the working nodes (servers) that compose the cluster, and to receive updates should any nodes be modified, added, or removed
- Watches the `Namespaces` resource (`watch namespaces`) for its cluster, so it can identify the namespaces needed for finding Pods
- Watches the `Pods` resource for each namespace (`watch pods`), recording each event when `Pods` are created, modified, or deleted; and extracting:
  - Basic identifying information about the `Pod`, and runtime information such as when the `Pod` was started

- The images on which the containers used on the pods are based
- Resource constraints applied to containers
- Usage: when, how many, and for how long Pods are used
- Software-identifying annotations applied to Pods

  Note: *Annotations can be of arbitrary size, and may potentially include sensitive data. For this reason, the lightweight Kubernetes agent captures only the known set of annotations required by the IBM License Service:*

  - *productID*
  - *productName*
  - *productMetric*
  - *cloudpakId*
  - *cloudpakName*
  - *cloudpakVersion*
  - *productChargedContainers*
  - *productCloudpakRatio.*

- If ancestry is configured, the Kubernetes resources that own Pods for contextualization

  Tip: *Currently there is no support in the web interface of FlexNet Manager Suite for displaying the chains of ownership of Pods, for which reason it is not recommended that you enable this functionality at this time.*

All of the above subscriptions take place concurrently, relying on the event subscriptions provided by Kubernetes (that is, it does not poll the API on a preset interval).

Separately, if the `--ibm-licensing` flag (and its companion flags) have been set during installation (or edited in the `deployment.yaml` file), the lightweight Kubernetes agent also collects data from the IBM License Service about IBM software (in particular, IBM Cloud Paks) running in the cluster, assembling a rolling window of 180 days of relevant licensing data (more details in Inventory from IBM License Service).

By default, the lightweight Kubernetes agent spends about 5 minutes gathering this data, and then writes it into one or more `.ndi` inventory files (see full details in Inventory Uploaded by the Kubernetes Agents and its sub-topics), and uploads the result to its nominated inventory beacon. It then waits for 24 hours (by default), and again writes the latest collected inventory into `.ndi` files for upload.

Downloading the Lightweight Kubernetes Agent

FlexNet Manager Suite 2023 R2 (On-Premises)

Both the Flexera Kubernetes inventory agent and the lightweight Kubernetes agent are available in a common tar archive.

The lightweight Kubernetes agent is deployed as a (Kubernetes resource) `Deployment` within the `flexera` namespace. The Pods run under a specific service account within that namespace, to which a specific custom cluster role is bound. The cluster role defines what actions the lightweight Kubernetes agent is allowed to perform.

To download and prepare the lightweight Kubernetes agent:

1. Log on to a device that:
   - Runs a supported version of Linux
   - Has a web browser with network access to the web application server for FlexNet Manager Suite, where your account must have operator privileges to see the appropriate page
   - Has a running instance of Docker
   - Either hosts an OCI container registry, or has network access to an OCI container registry, that is available to your Kubernetes cluster.

   Tip: *This process is simplified if you log in to this device using an account that has administrative privileges for your OCI container registry (otherwise you need to hand off between separate accounts during the process); and also for the Kubernetes cluster. During installation, the account requires privileges to create the following resource types:*
   - *`Namespace`*
   - *`ServiceAccount`*
   - *`ClusterRole`*
   - *`ClusterRoleBinding`*
   - *`Deployment`.*
2. In your web browser, navigate in FlexNet Manager Suite to **Discovery & Inventory > Settings**.

   The **Inventory Settings** page displays.
3. Expand the **Inventory agent for download** section.
4. Click **Download Flexera Kubernetes inventory agent** and save the downloaded archive file to a suitable location.

   Tip: *Despite the name on the link, the downloaded archive also includes the lightweight Kubernetes agent.*
5. Extract the folders from the downloaded archive, for example with the following command line:

   ```
   tar xzf flexera-krm-operator.tar.gz
   ```
6. Move into the directory that was extracted from the archive:

   Replace the placeholder *`x.y.z`* with the version number of the first extracted folder, and replace *`a.b.c`* with the version number included for the `lwk` folder for the lightweight Kubernetes agent.

   Tip: *These version numbers may not be the same. It may be easier to do this in two steps, where you can check*

the version number in the folder name.

```
cd flexera-krm-operator-x.y.z
cd lwk-a.b.c
```

This last folder contains the exported container image for the lightweight Kubernetes agent (as an exported tar archive, named `flexera-lwk-a.b.c.tar` where *a.b.c* is the version of the lightweight Kubernetes agent), the YAML resources that define the application within Kubernetes, and an (optional) installation shell script.

As the lightweight Kubernetes agent container image is not yet available as a Docker registry, a private registry available to the cluster must be used.

7. Import the extracted image into Docker:

   Replace *a.b.c* with the version number of the tar archive in the folder:

   ```
   docker load < flexera-lwk-a.b.c.tar
   ```

8. Re-tag the image for your registry.

   ***Tip:** If you have multiple container registries, complete the process for each one, and then circle back to repeat the process from this point for the next registry.*

   In these examples, the container registry is shown as *registry.example.org*, which you replace with the URL of your own registry. Also replace the placeholder *a.b.c* with the version number of the lightweight Kubernetes agent.

   ***Important:** The portion of the image name `flexera/lwk` must not be changed. Simply prepend your registry URL in front of this string as shown.*

   ```
   docker tag flexera/lwk:a.b.c registry.example.org/flexera/lwk:a.b.c
   ```

9. Ensure that you are logged into your OCI container registry, using an account with administrative privileges.

10. Push the image for lightweight Kubernetes agent to your registry:

    ```
    docker push registry.example.org/flexera/lwk:a.b.c
    ```

11. Inspect the YAML documents (supplied in the `install` directory) that are applied during installation and modify the Deployment.

    All of the RBAC resources that the lightweight Kubernetes agent uses are contained within the `rbac.yaml` file. You may also want to inspect the default `deployment.yaml` file (particularly if you plan to complete a manual installation process).

12. Choose between:

    - **Scripted installation** — This can automate the configuration of YAML resources without manual editing. Allows previewing (and optionally saving) the configuration before applying it. Requires the `kubectl` tool present on your working device when applying the resources to the cluster. For details, see Scripted Installation.

- **Manual installation** — Requires that you manually edit the `deployment.yaml` file, and then apply all required resources to the cluster (also using the `kubectl` tool). For details, see Manual Installation.

Whichever process you choose, the installation creates the following resources in the cluster.

Namespace flexera

```
apiVersion: v1
kind: Namespace
metadata:
  name: flexera
```

ServiceAccount flexera/lwk

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: lwk
  namespace: flexera
```

ClusterRole flexera-lwk

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: flexera-lwk
```

ClusterRoleBinding flexera-lwk

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: flexera-lwk
```

Deployment flexera/lwk

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: lwk
  namespace: flexera
```

Scripted Installation

In your downloaded archive where you extracted the lightweight Kubernetes agent (see Downloading the Lightweight Kubernetes Agent), the `install` sub-directory includes a Bash script, `install.sh`, that automates the installation process.

Tip: *Detailed usage information is available by running the install script with the `--help` flag.*

```
./install/install.sh --help
```

More information about available flags is available below, after the procedure.

To use the installation script:

1. Optionally, in your preferred flat text editor, update the `deployment.yaml` file with (at least) the registry where you pushed the container image, and the URL for the inventory beacon.

   Note: *If you wish to turn on monitoring so that the lightweight Kubernetes agent can expose Prometheus metrics on an HTTP endpoint, you must set the `--metrics` flag when you run the install script. This is an example of a flag that is not recognized by the install script itself, and, on the working assumption that this is an option to the `lwk` binary (lightweight Kubernetes agent) within the container, the install script appends it to the `args` attribute of the container.*

   For more detailed guidance about editing this file, see step 1 in Manual Installation. Editing this `deployment.yaml` file is particularly helpful when you also want to configure additional options for the lightweight Kubernetes agent, as described in Options for the Lightweight Kubernetes Agent. However, if you want only the mandatory changes (registry URL and inventory beacon URL), without monitoring, these can be included as flags for the installer, and it is then not necessary to edit the `deployment.yaml` file at all.

   Note: *If the lightweight Kubernetes agent is to upload to the inventory beacon using the HTTPS protocol, communications must be secured with TLS. In this case, it is necessary to edit the `deployment.yaml` file, as there are no installation flags available for TLS certificate management. For details, divert now to Managing Certificates for TLS, and return here after correctly configuring for the CA certificate bundle.*

2. Run the installation script with the appropriate flags.

   If you want to inspect the results of the configuration, include the `--stdout` flag (see flags listed below).

   - If you have configured all your details in the `deployment.yaml` file (with the updated file saved in place in the `install` sub-directory), simply run the Bash script for pre-inspection of the results:

     ```
     ./install/install.sh --stdout
     ```

     Or run the Bash script to configure the cluster with the settings in the `deployment.yaml` file:

     ```
     ./install/install.sh
     ```

   - If you are not using the `deployment.yaml` file for configuration, include the mandatory flags — whether for pre-inspection of the results (all on one line):

     ```
     ./install/install.sh --registry registry.example.org
                          --beacon https://beacon.example.org
                          --inventory-interval 6h
                          --stdout > configured.yaml
     ```

     Or to configure the cluster (all on one line):

```
./install/install.sh --registry registry.example.org
                     --beacon https://beacon.example.org
                     --inventory-interval 6h
```

Tip: *`--inventory-interval` is an example of a flag that is not recognized by the install script, and is therefore passed through to the lightweight Kubernetes agent. Details of options handled by the lightweight Kubernetes agent are included in Options for the Lightweight Kubernetes Agent.*

- If you want to manually specify a name for the Kubernetes cluster where the lightweight Kubernetes agent is being installed, include the `--cluster-name` flag:

```
./install/install.sh --cluster-name myorg-cluster-foo ...
```

Very shortly, Kubernetes instantiates your container, and the lightweight Kubernetes agent immediately begins gathering inventory about the `Node`, `Namespace`, and `Pod` resources in the cluster. By default, about 5 minutes later, the lightweight Kubernetes agent uploads the result to its nominated inventory beacon, by default to the path `%CommonAppData%\Flexera Software\Incoming\Inventories` (notice that files do not stay long in this folder, but are uploaded to the parent device in the hierarchy of inventory beacons, or to the central application server, as appropriate; but perhaps the last modified time-stamp on that folder gives some indication of work in progress). The lightweight Kubernetes agent then waits for a default 24 hours before repeating the process. (Modify these default cycle timings with the `--inventory-backoff` and `--inventory-interval` options, as described in Options for the Lightweight Kubernetes Agent.)

Flags for the install script

The install script accepts two types of command-line flags:

1. Flags that are directly handled by the script itself, either controlling the installation process, or causing the YAML resources to be modified.
2. Any flags that are not recognized by the install script are passed through directly to the lightweight Kubernetes agent, or in selected cases to the `kubectl` tool.

Flags handled by the install script are listed here in alphabetical order. Placeholders are shown *`thus`*:

`--as`	Optional, used only if the `kubectl` tool has not been correctly configured for the appropriate cluster. This flag is passed directly through to `kubectl`, so refer to its documentation online for details.

`--beacon`	Required by the lightweight Kubernetes agent, and so is mandatory either for the install script, or for one of these environment variables: • If the `--beacon` flag is not set, the install script checks for an environment variable called `LWK_BEACON`, and uses the value if this is found. • If the `--beacon` flag is not set and `LWK_BEACON` is not found, the install script checks for an environment variable called `BEACON`, and uses the value if this is found. If none of the above provide a URL to access the inventory beacon, the install script exits with an error. When a valid URL is obtained by any of the three possible methods, the install script validates that an inventory beacon has been provided, and then appends the value to the `args` attribute of the container for use by the lightweight Kubernetes agent when it is time to upload collected inventory. **Example:** `--beacon https://beacon.example.org`
`--cluster`	Optional, used only if the `kubectl` tool has not been correctly configured for the appropriate cluster. This flag is passed directly through to `kubectl`, so refer to its documentation online for details.
`--cluster-name`	Optional, use if you want to manually specify a name for the cluster where the lightweight Kubernetes agent is to operate. Because Kubernetes does not have a standard way to store names for its clusters, you may find that a manually-specified name is more meaningful.
`--context`	Optional, used only if the `kubectl` tool has not been correctly configured for the appropriate cluster. This flag and its value is added to the `kubectl` commands that the install script invokes, so refer to its documentation online for details.

`--extensions`	Accepts a comma-separated list (without spaces) of extensions that should be installed along with the lightweight Kubernetes agent. Extensions support optional features that need additional cluster permissions, or leverage third-party software that may (or may not) be installed in the cluster. They are defined by `.yaml` files found in the `install/extensions` subdirectory of the downloaded archive. Available extensions are: • `ancestry` — For future expansion, and not currently supported in the web interface of FlexNet Manager Suite. As defined in the `ancestry.yaml` file, this: ◦ Creates a new `ClusterRole` named `flexera-lwk-ancestry` ◦ Uses a `ClusterRoleBinding` to bind that role to the service account running the lightweight Kubernetes agent. ***Tip:*** *The service account called `lwk` is created in the `flexera` namespace by either installation method (see the list of resources created in the cluster given in Downloading the Lightweight Kubernetes Agent).* The new role gives permission for the lightweight Kubernetes agent to read (using the `get` verb) additional resource types that are known to be part of the ownership hierarchy of a `Pod`, such as a `ReplicaSet` or a `Deployment`, and to collect identifying information (`name`, `namespace`, `UID`, `type`.) It is not recommended that you enable this extension until there is support for displaying ancestry within the web interface. • `prometheus-service` — When its `--metrics` flag is set, the lightweight Kubernetes agent supports exposing Prometheus metrics using an HTTP endpoint. You can create a `Service` to expose the metrics endpoint widely within the cluster, or outside of the cluster. The `prometheus-service` extension, defined in the `prometheus-service.yaml` file, creates this `Service` using values that are valid for the default configuration of the lightweight Kubernetes agent. If the `--metrics` flag is not set, this extension is ignored. ***Tip:*** *To set the `--metrics` flag, it must be included when invoking the install script for the lightweight Kubernetes agent (see example below). The install script appends the flag to the `args` attribute of the container for use by the lightweight Kubernetes agent.* • `prometheus-servicemonitor` — When Prometheus is installed in the cluster using `prometheus-operator` (see https://github.com/prometheus-operator/prometheus-operator), and Prometheus metrics are enabled on the lightweight Kubernetes agent (that is, the `--metrics` flag is set and the `prometheus-service` extension is configured), the `prometheus-servicemonitor` extension creates a `ServiceMonitor`, the custom resource type used by `prometheus-operator` to automatically configure Prometheus to scrape a metrics endpoint.

This `ServiceMonitor` allows Prometheus to automatically begin scraping metrics for the lightweight Kubernetes agent.

Important: *Use only in conjunction with the `prometheus-service` extension.*

Example: (flags have been wrapped onto separate lines for presentation)

```
./install/install.sh
    --registry registry.example.org
    --beacon https://beacon.example.org
    --metrics
    --extensions prometheus-service,prometheus-servicemonitor
```

`--help`	Must be used alone on the command line for the install script. Prints usage information to the screen. **Example:** `./install/install.sh --help`
`--kubeconfig`	Optional, used only if the `kubectl` tool has not been correctly configured for the appropriate cluster. This flag is passed directly through to `kubectl`, so refer to its documentation online for details.
`--registry`	Specify the Docker image registry to which the container image was pushed (see Downloading the Lightweight Kubernetes Agent). The install script injects the value of this flag into the `image` attribute of the container. ***Note:*** *Do not use this flag if the `Deployment` in the `deployment.yaml` file has been edited to add the registry. This flag is available as an alternative, not a duplicate.* **Example:** `--registry registry.example.org`
`--stdout`	Redirects the YAML resource so that it is *not* applied to the cluster, and is instead printed to the screen. You may optionally add shell redirection to save the configured result to a file for saving or sharing. **Example:** (all on one line) `./install/install.sh --registry registry.example.org --beacon https://beacon.example.org --stdout > configured.yaml`

`--uninstall`	Not relevant to the installation process, obviously. For details, see Uninstalling the Lightweight Kubernetes Agent.
`--version`	It is best practice not to use this flag. The version of the lightweight Kubernetes agent is set in the `image` tag for the `Deployment` resource in the `deployment.yaml` file. In rare cases, you can specify a different version of the lightweight Kubernetes agent, overriding the setting in the `Deployment` resource; but best practice, if a different version of the lightweight Kubernetes agent should be installed, is to download that version and use its install script instead. ***Tip:*** *It is not possible to change the standard image name, `flexera/lwk`, using the install script.* **Example:** `--version a.b.c`

Manual Installation

Manual installation of the lightweight Kubernetes agent is straight-forward: simply modify the YAML resources to suit your needs, and then apply them to the cluster.

Note: *This process assumes that you have completed downloading and deploying the lightweight Kubernetes agent, as described in Downloading the Lightweight Kubernetes Agent.*

This process requires the `kubectl` tool installed on your working device.

To manually install the lightweight Kubernetes agent:

1. In your preferred flat text editor, update the `deployment.yaml` file as follows:

   Mandatory changes:

   a. Update the `image` setting in the `Deployment` to specify the registry where you pushed the container image (for details, see Downloading the Lightweight Kubernetes Agent). In the following example, replace the placeholder *`registry.example.org`* with the URL of your own registry; and replace the placeholder *`a.b.c`* with the version number of the lightweight Kubernetes agent.

   b. Set the `--beacon` flag to the URL of the inventory beacon to which the lightweight Kubernetes agent will upload its collected inventories (substituting your own value for the placeholder).

   c. During operation of the lightweight Kubernetes agent, do you want it to expose Prometheus telemetry metrics so that you can monitor performance? If so, it is mandatory to set the `--metrics` flag in the YAML file, as shown in this example:

```
apiVersion: apps/v1
kind: Deployment
```

```
...
spec:
  template:
    spec:
      containers:
        - name: agent
          image: registry.example.org/flexera/lwk:a.b.c
          args:
            - --beacon
            - https://beacon.example.org
            - --metrics
```

Note: *If the lightweight Kubernetes agent is to upload to the inventory beacon using the HTTPS protocol, communications must be secured with TLS. In this case, additional edits are required in the* `deployment.yaml` *file. For details, divert now to Managing Certificates for TLS, and return here after correctly configuring for the CA certificate bundle.*

Optional changes: Add any other configuration flags that you require to the `Deployment` in a similar fashion. These may be labels or annotations, security context configuration, or resource limits. For details of the options that can be specified as configuration flags, see Options for the Lightweight Kubernetes Agent.

When your changes are complete, save the updated file in place.

2. Apply the resources to the cluster, for example using the following command lines (which here include the optional extensions, discussed below):

```
cd install
kubectl apply -f namespace.yaml
kubectl apply -f rbac.yaml
kubectl apply -f deployment.yaml
kubectl apply -f extensions/prometheus-service.yaml
kubectl apply -f extensions/prometheus-servicemonitor.yaml
```

Very shortly, Kubernetes instantiates your container, and the lightweight Kubernetes agent immediately begins gathering inventory about the `Node`, `Namespace`, and `Pod` resources in the cluster. By default, about 5 minutes later, the lightweight Kubernetes agent uploads the result to its nominated inventory beacon, by default to the path `%CommonAppData%\Flexera Software\Incoming\Inventories` (notice that files do not stay long in this folder, but are uploaded to the parent device in the hierarchy of inventory beacons, or to the central application server, as appropriate; but perhaps the last modified time-stamp on that folder gives some indication of work in progress). The lightweight Kubernetes agent then waits for a default 24 hours before repeating the process. (Modify these default cycle timings with the `--inventory-backoff` and `--inventory-interval` options, as described in Options for the Lightweight Kubernetes Agent.)

Extensions

Extensions support optional features that need additional cluster permissions, or leverage third-party software that may (or may not) be installed in the cluster. They are defined by `.yaml` files found in the `install/extensions` subdirectory of the downloaded archive. Available extensions are:

- `ancestry` — For future expansion, and not currently supported in the web interface of FlexNet Manager Suite. As

defined in the `ancestry.yaml` file, this:

- Creates a new `ClusterRole` named `flexera-lwk-ancestry`
- Uses a `ClusterRoleBinding` to bind that role to the service account running the lightweight Kubernetes agent.

Tip: *The service account called `lwk` is created in the `flexera` namespace by either installation method (see the list of resources created in the cluster given in Downloading the Lightweight Kubernetes Agent).*

The new role gives permission for the lightweight Kubernetes agent to read (using the `get` verb) additional resource types that are known to be part of the ownership hierarchy of a `Pod`, such as a `ReplicaSet` or a `Deployment`, and to collect identifying information (`name`, `namespace`, `UID`, `type`.) It is not recommended that you enable this extension until there is support for displaying ancestry within the web interface.

- `prometheus-service` — When its `--metrics` flag is set, the lightweight Kubernetes agent supports exposing Prometheus metrics using an HTTP endpoint. You can create a `Service` to expose the metrics endpoint widely within the cluster, or outside of the cluster. The `prometheus-service` extension, defined in the `prometheus-service.yaml` file, creates this `Service` using values that are valid for the default configuration of the lightweight Kubernetes agent. If the `--metrics` flag is not set, this extension is ignored.

Tip: *To set the `--metrics` flag, it must be included when invoking the install script for the lightweight Kubernetes agent (see example below). The install script appends the flag to the `args` attribute of the container for use by the lightweight Kubernetes agent.*

- `prometheus-servicemonitor` — When Prometheus is installed in the cluster using `prometheus-operator` (see https://github.com/prometheus-operator/prometheus-operator), and Prometheus metrics are enabled on the lightweight Kubernetes agent (that is, the `--metrics` flag is set and the `prometheus-service` extension is configured), the `prometheus-servicemonitor` extension creates a `ServiceMonitor`, the custom resource type used by `prometheus-operator` to automatically configure Prometheus to scrape a metrics endpoint. This `ServiceMonitor` allows Prometheus to automatically begin scraping metrics for the lightweight Kubernetes agent.

Important: *Use only in conjunction with the `prometheus-service` extension.*

Managing Certificates for TLS

On a regular interval (by default, once every 24 hours, or the setting you provide through the `--inventory-interval` flag), the lightweight Kubernetes agent uploads its collected inventory to a single inventory beacon, for which the URL is specified in the `--beacon` flag or alternative environment variables (for details, see Options for the Lightweight Kubernetes Agent). If this URL is not specified, the lightweight Kubernetes agent immediately exits with an error printed to `stdout`, and viewable with the following command (replacing the placeholder with the appropriate pod name):

```
kubectl logs -n flexera lwkPod
```

As always, the complete URL must include the protocol (HTTP or HTTPS). If your inventory beacon is configured for HTTPS communication, secure communications are protected with Transport Layer Security (TLS).

Tip: *Currently the lightweight Kubernetes agent supports only standard (single-sided) TLS, and does not support mutual TLS.*

TLS requires that the inventory beacon presents a server certificate that can be validated by the client (in this case, the lightweight Kubernetes agent) against a certificate chain culminating in a root certificate for the issuing Certificate Authority (CA). This means that the CA root certificate (and any intermediate certificates, as applicable) must be present and accessible on the client device. Otherwise, the lightweight Kubernetes agent will refuse the connection to the inventory beacon because of an untrusted certificate (but see also `--ibm-licensing-tls-verify` inOptions for the Lightweight Kubernetes Agent).

To validate the server certificate presented by the inventory beacon, the lightweight Kubernetes agent checks for a file mounted into the container at the path `/beacon.pem`. If this is found, the certificates it contains are appended to the trust bundle for the lightweight Kubernetes agent.

Certificate bundle constraints

- The CA certificate bundle *must* use the PEM encoding.
- The server certificate used by the inventory beacon *must* use the `Subject Alternative Name` extension. (This is because the lightweight Kubernetes agent is implemented in Go [version 1.16 or later]. After Go version 1.5, clients no longer support server identification using the `Common Name` attribute of the certificate.) If your current certificate for the relevant inventory beacon does not currently include the `Subject Alternative Name`, you need to generate a new CSR, with the `Common Name` attribute the same as in the current certificate, and reissue the certificate.

To configure the lightweight Kubernetes agent for a custom CA certificate bundle:

1. If necessary, create a new Certificate Signing Request (CSR) for a CA to supply you with a new certificate that includes the `Subject Alternative Name`.

   For a reminder about how to prepare a CSR, see the online help under **FlexNet Manager Suite Help > Inventory Beacons > Local Web Server Page > Configuring Mutual TLS** at step 7, remembering to keep the same `Common name` as in the current certificate, and to add the `Subject Alternative Name` (ignore the fact that the help page is about mutual TLS, since the process of preparing a CSR is the same).

2. If necessary, when a replacement certificate is received, load it into the inventory beacon.

   For a process reminder, see steps 8 and 10 in the same help topic.

3. Take a copy of the CA root certificate from the inventory beacon, convert it to the `.pem` format, and save as `ca-certificates.pem`.

   One method is to use the `openssl` toolkit, available through https://www.openssl.org/source/, on a convenient Windows device where you have `openssl` and a copy of the `.pfx` file you are deploying for Windows devices.

   a. To export the certificate (including the necessary public key) in a `.pem`file:

   ```
   openssl pkcs12 -in filename.pfx -clcerts -nokeys -out ca-certificates.pem
   ```

b. Open the resulting certificate file in your preferred flat text editor (such as Notepad), delete all preliminary lines of text before `-----BEGIN CERTIFICATE-----`, and save the amended file.

c. If the certificate conversion has been completed on a different computer (such as, perhaps, a Windows-based inventory beacon), copy the finished `.pem` file to your working Linux-based computer.

4. Add the CA certificate bundle to the Kubernetes `ConfigMap` (from your Linux-based computer):

```
kubectl create configmap custom-certs -n flexera --from-file=ca-certificates.pem
```

5. In your preferred flat text editor, edit the `deployment.yaml` file to identify your `configMap` and define the appropriate storage.

   For example, if your `configMap` is named `custom-certs` (in the `flexera` namespace), you can name it as a volume `beacon-ca-certificate` in the `pod` section of your `deployment.yaml` file for lightweight Kubernetes agent:

```
apiVersion: apps/v1
kind: Deployment
...
spec:
  template:
    spec:
      volumes:
        - name: beacon-ca-certificate
          configMap:
            name: custom-certs
      containers:
        - name: agent
          volumeMounts:
            - name: beacon-ca-certificate
              mountPath: /beacon.pem
              subPath: ca-certificate.pem
              readOnly: true
```

Tip: *The names of the `configMap` and the `volume` are not significant, and you may customize the names to suit your environment. However, the `mountPath` element of the `volumeMount` **must** be set to `/beacon.pem`.*

Now, with your `deployment.yaml` file customized for the CA certificate bundle, you can resume your installation process, whether it is Scripted Installation or Manual Installation.

Uninstalling the Lightweight Kubernetes Agent

If you wish to remove the lightweight Kubernetes agent from your cluster, there are two alternative paths: you may use the install script, or allow Kubernetes to do the removal by removing the YAML resources.

Note: *Neither of these methods removes the `flexera` namespace, to ensure that any other resources within the*

namespace are not inadvertently deleted. As well, you may intend to add other Flexera components back into the namespace — for example, you may be removing a superseded version of the lightweight Kubernetes agent and replacing it with a later release.

To uninstall the lightweight Kubernetes agent:

1. Choose your preferred method:
   - **Install script:** On the appropriate Linux-based device, run the install script with the `--uninstall` flag. The only other flags relevant to the uninstall process are:
     - `--as`
     - `--cluster`
     - `--context`
     - `--kubeconfig`.

     For information about these flags, see the latter part of Scripted Installation.

     ```
     ./install/install.sh --uninstall
     ```

     ***Note:** The install script also attempts to uninstall the `prometheus-servicemonitor` extension. If the `prometheus-operator` is not installed in the cluster, this results in an error message, as the cluster will not have a `ServiceMonitor` type. In these circumstances, this error can be safely ignored.*
   - **kubectl:** On the appropriate Linux-based device, use the `kubectl delete` command to remove the YAML files, after which Kubernetes automatically removes the deployment from the cluster:

     ```
     cd install
     kubectl delete -f deployment.yaml
     kubectl delete -f rbac.yaml
     ```
2. Optionally, remove the `flexera` namespace (if you will not be re-using that in future):

   ```
   kubectl delete namespace flexera
   ```

Options for the Lightweight Kubernetes Agent

Alternative modes

These first two flags are not operational flags, but if you require more information, these may be set on the command line for the lightweight Kubernetes agent when run underneath Docker:

`--version`	Prints the long-form version of the lightweight Kubernetes agent to `stdout`, and exits. `docker run --rm flexera/lwk:x.y.z --version`
`--help`	Prints the built-in usage information to `stdout`, and exits. `docker run --rm flexera/lwk:x.y.z --help`

For normal operation, the following flags may be set in the `deployment.yaml` file that modifies the Deployment during installation.

Duration values

Flags that accept a duration of time use the formatting from the Go programming language: a numeric value, followed immediately by a unit suffix: `h` for hours, `m` for minutes, or `s` for seconds. Units larger than a hour are not supported, so days or weeks must be specified by calculating the number of hours. For example:

- 24 hours is `24h`
- 15 minutes is `15m`
- 30 seconds is `30s`
- 2 days is `48h`
- 1 week (7 days) is `168h`.

Option values

Placeholders for values to supply are shown *`thus`*.

***Tip:** Flags that require a Boolean value (true, false) **must** have an equals sign (=) linking the flag and value. All other flags may have a space between the flag name and value.*

Available flags (listed in alphabetical order) include:

`--beacon URL`

Type: Valid URL (**required** — otherwise the lightweight Kubernetes agent will log an error and exit if this is not specified)

Default: Unset

The URL of an inventory beacon to which collected inventories are uploaded every 24 hours (by default, or see `--inventory-interval`). Must include the protocol (either `http://` or `https://`), the host name, an optional port number (if used, separated from the host name by a colon), and any necessary path elements to reliably reach the inventory beacon. The value may include any or none of the `/ManageSoftRL/Inventory` path components used within the inventory beacon — these will be automatically appended by the lightweight Kubernetes agent if omitted from the flag (as in the example below).

Tip: *Uptime on the inventory beacon, and network reliability on the path from the lightweight Kubernetes agent to the inventory beacon, are critical to inventory gathering. Because the lightweight Kubernetes agent does not install any components of the FlexNet inventory agent, it does not include the inventory beacon failover, or overnight catch-up uploads to recover from temporary network outages. The inventory beacon specified with this flag is the only one used by the lightweight Kubernetes agent. Also note that if you are using the `https` protocol, the lightweight Kubernetes agent supports only standard TLS to authenticate those communications.*

Example:

```
--beacon https://mybeacn.example.com:443/leaveK8s
```

`--exclude-namespace name`

`--exclude-namespaces name,name,...`

Type: Kubernetes text (lower-case alphanumeric characters and dashes), with list comma-separated without whitespace characters or quotation marks

Default: Unset

Add one or more namespaces to the exclusion list in the namespace filter used by the lightweight Kubernetes agent:

- The `--exclude-namespace` flag accepts only a single name, but may be specified any number of times. All of the occurrences of the flag are combined into a list.
- The `--exclude-namespaces` flag accepts a comma-separated list of namespaces, and can only be specified once.

The two flags are additive. All of the instances of `--exclude-namespace` are combined with the value of `--exclude-namespaces`, and the resulting list of namespaces is deduplicated.

Examples:

```
--exclude-namespace private
```

```
--exclude-namespaces private,testing
```

`--ibm-licensing`

Type: Boolean

Default: false (when unspecified, this default is provided in the code of the lightweight Kubernetes agent)

A Boolean option that, when true, enables integration with the IBM License Service. This is means that, by default, the lightweight Kubernetes agent *cannot* gather licensing data for IBM Cloud Paks running in Kubernetes containers (recall that using the IBM License Service is mandatory to meet IBM's license terms in this environment). You must explicitly enable this integration.

The examples below show the two ways of enabling the integration, either by:

- Specifying the flag with no assigned value
- Specify the flag with an explicit Boolean value.

Important: *When this flag is set to true by either method, the other flags pertaining to integration with the IBM License Service integration are mandatory. These flags all begin with `ibm-licensing`… If this flag is set and either of `--ibm-licensing-url` or `--ibm-licensing-token` is missing, the lightweight Kubernetes agent aborts with an error.*

Examples:

```
--ibm-licensing=true
--ibm-licensing
```

`--ibm-licensing-tls-verify`

Type: Boolean string

Default: True

By default, the lightweight Kubernetes agent properly verifies the server certificate for TLS certification of HTTPS communications with the IBM Licensing Service. However, if the IBM License Service is using an untrusted certificate, this option can be set to false to allow the lightweight Kubernetes agent to ignore certificate verification errors.

Tip: *Unless you supplied the IBM License Server with a certificate and key certified by a Certificate Authority, its default behavior is to use a self-signed certificate, and in that case you should turn off the formal verification process that looks for a root certificate from a Certificate Authority — as in the following example.*

Example:

```
--ibm-licensing-tls-verify=false
```

`--ibm-licensing-token` *`string`*

Type: String

Default: Unset (**required** when `--ibm-licensing true`)

The authentication token the lightweight Kubernetes agent can use to authenticate with the IBM License Service. This is typically stored in a `ConfigMap` in the cluster. If the token is changed, the value of this flag must also be changed to match, and the lightweight Kubernetes agent must then be restarted.

Example:

```
--ibm-licensing-token jNsshfn6scWre4unzCWlO7xs
```

`--ibm-licensing-url` *`url`*

Type: Valid URL

Default: Unset (**required** when `--ibm-licensing true`)

The complete URL, including the protocol, the host name, an optional port number (if used, separated from the host name by a colon), and any necessary path elements to reliably reach the IBM License Service.

Example:

```
--ibm-licensing-url

https://ibm-licensing-service-instance.ibm-common-services.svc:8080
```

`--include-namespace` *`name`*

`--include-namespaces` *`name,name,...`*

Type: Kubernetes text (lower-case alphanumeric characters and hyphens), with list comma-separated without whitespace characters or quotation marks

Default: Unset

Add one or more namespaces to the inclusion list in the namespace filter used by the lightweight Kubernetes agent:

- The `--include-namespace` flag accepts only a single name, but may be specified any number of times. All of the occurrences of the flag are combined into a list.
- The `--include-namespaces` flag accepts a comma-separated list of namespaces, and can only be specified once.

The two flags are additive. All of the instances of `--include-namespace` are combined with the value of `--include-namespaces`, and the resulting list of namespaces is deduplicated.

Examples:

```
--include-namespace production
```

```
--include-namespaces production,testing
```

`--inventory-backoff` *duration*	*Type:* Duration value (see details above) *Default:* `5m` Set the duration of time for which the lightweight Kubernetes agent gathers data before uploading the first inventory. Unlike the standard FlexNet inventory agent, the lightweight Kubernetes agent does not use a polling strategy to gather inventory. Instead, it continuously monitors the state of the cluster by consuming event streams. In theory, it can upload an inventory file at any point after it connects to Kubernetes and begins receiving data. However, if the lightweight Kubernetes agentuploads too soon, it may not have received the initial state of each resource in the cluster, leaving the first inventory incomplete (and by default, this would only be corrected a day later). The default value is 5 minutes, which is sufficient to receive all of the data for nearly all clusters. For very small clusters or for testing purposes, this value can be shortened to mere seconds. *Example:* `--inventory-backoff 10m`
`--inventory-interval` *duration*	*Type:* Duration value (see details above) *Default:* Unset (in which case the lightweight Kubernetes agent uses a value of `24h`) Set the time interval on which the lightweight Kubernetes agent collects and uploads inventories. General best practice is to omit this flag, accepting the default of 24 hours, as a shorter interval requires uploading and processing a larger number of smaller inventories. However, you may adjust this interval to meet the needs of your enterprise for frequency of inventory gathering. *Example:* `--inventory-interval 48h`

`--label-selector` *selector*

`--namespace-label-selector` *selector*

`--node-label-selector` *selector*

`--pod-label-selector` *selector*

Type: Kubernetes label selector (using either equality-based requirement or [within single quotes] set-based requirement, see https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/)

Default: Unset

Set the label selectors that are used when subscribing to the event streams in the Kubernetes API. A distinct selector, as indicated by the names, can be applied to the subscription to each of the Namespace, Node, and Pod resources. The general `--label-selector` flag, if specified, is used for whichever of the three distinct subscriptions have not been specified. Only events that match all of the specified selectors are used to add to the inventory for upload.

In the following example, inventory is calculated from events coming from pods where the `environment` label is set to one of the following:

```
environment: qa
environment: production
```

and the namespace and node both have a label set:

```
app: store
```

Example:

```
--pod-label-selector 'environment in (production, qa)'
--label-selector app=store
```

`--log-level level`

Type: String option list

Default: `info`

Set the logging verbosity level to one of:

- `info` — Provides all of the information that is typically useful
- `debug` — Available for additional information or troubleshooting, and produces a high volume of information
- `trace` — Provides even more information than `debug`, and typically should not be needed
- `warn` — Only produce messages logged at warning level or higher (not recommended because, should an issue arise, this can mask valuable troubleshooting information)
- `error` — Only produce messages logged at error level or higher (not recommended because, should an issue arise, this can mask valuable troubleshooting information)
- `fatal` — Suppress all messages except fatal errors resulting in immediate termination of the application (not recommended because, should an issue arise, this can mask valuable troubleshooting information).

Other string values not in this list are ignored.

Example:

```
--log-level debug
```

`--metrics`

Type: Boolean string

Default: False

When true, this option enables an HTTP endpoint on the lightweight Kubernetes agent to serve Prometheus metrics. By default, the metrics endpoint is not enabled. As in the following examples, it can be enabled using the option flag by itself, or with an explicit boolean value,

Examples:

```
--metrics
--metrics=true
```

`--metrics-address host:port`	*Type:* String *Default:* `:9090` (matching the Prometheus default port) When Prometheus metrics are enabled using `--metrics`, this flag sets the host and port to which the HTTP server is bound. Using the default value where the host is not set (and the colon is mandatory to precede the port number), the default is to listen on the Pod's internal, automatically-assigned IP address, on the nominated port. *Example:* `--metrics-address :80`
`--metrics-endpoint path`	*Type:* String giving valid path with the metrics server *Default:* `/metrics` (supplied by internal code when flag is unset) When Prometheus metrics are enabled using `--metrics`, this flag sets the path on which the metrics service is mounted. This value can be appended to the value of `metrics-address` to complete the HTTP path. ***Tip:*** *If you change this value, the lightweight Kubernetes agent must be restarted.* *Example:* `--metrics-endpoint /prometheus`

`-v`	*Type: No value required*
	Default: Unset
	An alias for
	`--log-level debug`
	that provides a shorthand way to increase the logging verbosity level to `debug`.
	Example:
	`-v`

`--volume path`

Type: A valid path within the container's file system.

Default: Unset

By default, the lightweight Kubernetes agent assumes that the container is immutable at run-time and therefore does not have any persistent storage volumes mounted. As a result, by default:

- All collected inventory data is cached in memory
- Inventory is never written to a file on the agent end of the process
- Run-time state is lost on restarts.

Note: *Some of the Kubernetes libraries that are utilized by the lightweight Kubernetes agent (notably the Kubernetes client) may attempt to interact with the file system, such as reading random data from the random device node, or reading the service account token that is mounted into the container. With these exceptions, the lightweight Kubernetes agent behaves by default as if there were no file system.*

However, you may change that default behavior by providing a path in this `--volume` flag. A value here instructs the lightweight Kubernetes agent that there is a persistent, mutable volume mounted at the specified directory. It then uses the volume to cache its inventory data in that directory on disk. This reduces the memory overhead of the lightweight Kubernetes agent by allowing it to cache to disk instead of in memory.

The most common pattern for providing the agent with persistent storage, and the method supported by the lightweight Kubernetes agent, is to first create a `PersistentVolumeClaim` (for example, see https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/). When the `--volume` flag exists in the `deployment.yaml` file, the agent's `Deployment` is modified to mount the volume provided by the `PersistentVolumeClaim` to a path in the container's file system. Finally, the `--volume` flag is added to the command line for the lightweight Kubernetes agent in the `Deployment`, along with the path in the container file system where the volume was mounted.

Example:

```
--volume /data
```

Advanced options

These options rarely require modification, have a significant impact on the behavior of the lightweight Kubernetes agent, or are present to support future expansion.

`--cluster-interval duration`

Type: Duration value (see details above)

Default: `24h`

Sets the duration of time after which the lightweight Kubernetes agent will issue requests to update the basic metadata about the cluster. For clusters whose version rarely changes, the interval can be much wider to (slightly) reduce the number of requests to the Kubernetes API.

Example:

```
--cluster-interval 96h
```

`--connect-profile name`

Type: String

Sets the algorithm used to provision and configure the Kubernetes API client. Currently, this value must not be modified.

`--restore-mode mode`

Type: String option list

Default: `clean`

In the scenario where:

- File storage has been enabled using the `--volume` flag, and
- Collected data has been persisted to storage, and
- The lightweight Kubernetes agent is restarted

the agent must resolve changes that have occurred in the cluster since the data was written. This flag sets the behavior of the lightweight Kubernetes agent when an unrecoverable discrepancy has been detected, such as data referencing a different cluster. The options are:

- `clean` — The lightweight Kubernetes agent deletes the persisted data, and proceeds with normal operation using the current environment
- `refuse` — The lightweight Kubernetes agent leaves the persisted data unchanged, and exits with a failure.

Example:

```
----restore-mode refuse
```

2

Inventory Uploaded by the Kubernetes Agents

The Flexera Kubernetes inventory agent and the lightweight Kubernetes agent both produce inventory of the Kubernetes resources observed through the Kubernetes API, including `Nodes`, `Pods`, and `Namespaces`. Like all FlexNet inventory, the information is uploaded in `.ndi` files, and for Kubernetes inventory these use the following naming conventions (where *`clusterId`* is a short-form cluster ID):

Filename pattern	Description
`k8s-inventory-`*`clusterId-timestamp`*`.ndi`	Primary Kubernetes resource inventory
`k8s-ibm-licensing-`*`clusterId-timestamp`*`.ndi`	IBM License Service data

The full Flexera Kubernetes inventory agent (only) also uploads two additional `.ndi` files:

Filename pattern	Description (full agent only)
`k8s-image-`*`imageId`*`.ndi`	Container image software inventory
`k8s-node-`*`clusterId-nodeId-timestamp`*`.ndi`	Worker node hardware inventory

Through the Kubernetes API, either agent receives a series of events that occur on the resources it is monitoring in the cluster. Each event occurs because a resource has been modified, and the content of the event is the new state of the modified resource. Either agent extracts the data that is relevant for the resource (based on the resource type), and maintains that data in a local cache:

- Because the lightweight Kubernetes agent by default treats its container as immutable, it holds all its cached data in memory
- The full Flexera Kubernetes inventory agent has local storage available, and writes its cache to that.

Specifically, either agent:

- Read the `kube-system` `Namespace`, to obtain its UID to use as a cluster identifier (`get namespace`)
- Read the cluster's version

- Watch the `Nodes` resource (`watch nodes`)
- Watch the `Namespaces` resource (`watch namespaces`)
- Watch the `Pods` resource for each namespace (`watch pods`).

This process of event capture and caching occurs the entire time that either agent is running, but inventories are only produced on a set interval (by default, once every 24 hours). When the interval expires, either agent reads a snapshot of the cache, formats the data, and does one of two things:

- If it is the lightweight Kubernetes agent, *and* no storage volume has been declared using the `--volume` switch, the content is sent directly to the inventory beacon in an HTTP request
- In all other cases (that is, if you have assigned local storage for the lightweight Kubernetes agent, or if you are using the full Flexera Kubernetes inventory agent), the agent writes out the data in the appropriate inventory files to the persistent storage.

The `.ndi` files are uploaded to the inventory beacon as soon as they are completed.

The following topics detail the data uploaded in common by either of the Kubernetes agents. These consist of:

- The primary Kubernetes resource inventory, including `Nodes`, `Pods`, and `Namespaces` that are observed through the Kubernetes API, and reported as a series of data classes inside the `k8s-inventory-`*`clusterId-timestamp`*`.ndi` file (for details, see Kubernetes Inventory Uploads)
- Information collected from the IBM License Service, reported in `k8s-ibm-licensing-`*`clusterId-timestamp`*`.ndi` (for details, see Inventory from IBM License Service).

Kubernetes Inventory Uploads

The following data classes, each representing a particular Kubernetes resource type or other information about the cluster, are uploaded as part of the primary Kubernetes resource inventory in the `k8s-inventory-`*`clusterId-timestamp`*`.ndi` file.

MGS_KubernetesCluster

Metadata about the cluster itself, primarily the Kubernetes version. Example:

```
<Hardware Class="MGS_KubernetesCluster" Name="d8259158-fdfe-4ba3-ae28-be62a51df7f9"
Evidence="OS API">
    <Property Name="Name" Value="d8259158-fdfe-4ba3-ae28-be62a51df7f9"/>
    <Property Name="ID" Value="d8259158-fdfe-4ba3-ae28-be62a51df7f9"/>
    <Property Name="Version" Value="v1.21.2"/>
    <Property Name="MajorVersion" Value="1"/>
    <Property Name="MinorVersion" Value="21"/>
    <Property Name="Platform" Value="linux/amd64"/>
    <Property Name="BuildDate" Value="2021-06-16T12:53:14Z"/>
</Hardware>
```

Field	Description
Name	A descriptive name for the cluster
ID	Unique identifier for the cluster
Version	The full version of Kubernetes
MajorVersion	The major version of Kubernetes
MinorVersion	The minor version of Kubernetes
Platform	The operating system and CPU architecture for which the Kubernetes binaries were built
BuildDate	The date the Kubernetes binaries were built

MGS_KubernetesNode

Data representing a Node. Example:

```
<Hardware Class="MGS_KubernetesNode" Name="f97a7706-2890-4ac7-96ab-6d9405d81d1e"
Evidence="OS API">
    <Property Name="Name" Value="foo.example.org"/>
    <Property Name="UID" Value="f97a7706-2890-4ac7-96ab-6d9405d81d1e"/>
    <Property Name="ResourceVersion" Value="236788"/>
    <Property Name="Created" Value="2021-07-26T17:19:36Z"/>
    <Property Name="OS" Value="linux"/>
    <Property Name="Arch" Value="amd64"/>
    <Property Name="MachineID" Value="da8ad26e83c04fb5a2700f0299824737"/>
    <Property Name="KubeletVersion" Value="v1.21.2"/>
    <Property Name="Runtime" Value="docker://20.10.6"/>
    <Property Name="BootID" Value="0377939c-1adb-4ff6-afb3-fd71910719f6"/>
    <Property Name="CPUs" Value="16"/>
    <Property Name="MemoryBytes" Value="274530623488"/>
</Hardware>
```

Field	Description
Name	The name of the node, typically the hostname of the underlying server

Field	Description
UID	Unique identifier for the node
ResourceVersion	The cluster version at which the node was last modified
Created	The time at which the node was created
Deleted	The time at which the node was deleted
OS	The operating system of the node's underlying server
Arch	The CPU architecture of the node's underlying server
MachineID	A unique identifier passed through from the node's underlying server
KubeletVersion	The version of the Kubernetes kubelet component running on the node
Runtime	The container runtime used on the node
BootID	An identifier generated by the operating system on each boot
CPUs	CPU capacity of the node as a number of cores
MemoryBytes	Memory capacity of the node in bytes

MGS_KubernetesNamespace

Represents a Namespace in use within the cluster. Example (notice that the first line has been wrapped for presentation):

```
<Hardware Class="MGS_KubernetesNamespace" Name="d8259158-fdfe-4ba3-ae28-be62a51df7f9"
         Evidence="OS API">
    <Property Name="Name" Value="kube-system"/>
    <Property Name="UID" Value="d8259158-fdfe-4ba3-ae28-be62a51df7f9"/>
    <Property Name="ResourceVersion" Value="26"/>
    <Property Name="Created" Value="2021-07-26T17:19:36Z"/>
</Hardware>
```

Field	Description
Name	The name of the namespace
UID	Unique identifier for the namespace
ResourceVersion	The cluster version at which the namespace was last modified
Created	The time at which the namespace was created
Deleted	The time at which the namespace was deleted

MGS_KubernetesOwnerReference

Represents a non-Pod resource that is part of a chain of ownership beginning with a Pod. Contains the UID of its owner, if it has one. Examples (notice that the first lines of each class have been wrapped for presentation):

```
<Hardware Class="MGS_KubernetesOwnerReference"
Name="6b3b1428-572e-4cf8-aa6d-0672d5defc04"
        Evidence="OS API">
    <Property Name="APIVersion" Value="apps/v1"/>
    <Property Name="Kind" Value="ReplicaSet"/>
    <Property Name="Name" Value="coredns-558bd4d5db"/>
    <Property Name="UID" Value="6b3b1428-572e-4cf8-aa6d-0672d5defc04"/>
    <Property Name="Owner" Value="b5206863-bed7-477e-be40-be1031795c34"/>
</Hardware>
<Hardware Class="MGS_KubernetesOwnerReference"
Name="b5206863-bed7-477e-be40-be1031795c34"
        Evidence="OS API">
    <Property Name="APIVersion" Value="apps/v1"/>
    <Property Name="Kind" Value="Deployment"/>
    <Property Name="Name" Value="coredns"/>
    <Property Name="UID" Value="b5206863-bed7-477e-be40-be1031795c34"/>
</Hardware>
```

Field	Description
APIVersion	The API group and version of the resource's type
Kind	The resource type

Field	Description
Name	The name of the resource
UID	Unique identifier for the resource
Owner	The UID of the resource that owns this resource (property omitted when the owner is not known)

MGS_KubernetesPod

Represents a Pod. Example (notice that the first line has been wrapped for presentation):

```
<Hardware Class="MGS_KubernetesPod" Name="5c5a521b-7d58-47ca-8983-536bc2ec8c75"
         Evidence="OS API">
    <Property Name="Name" Value="coredns-558bd4d5db-4w2sc"/>
    <Property Name="Namespace" Value="kube-system"/>
    <Property Name="UID" Value="5c5a521b-7d58-47ca-8983-536bc2ec8c75"/>
    <Property Name="Created" Value="2021-07-26T17:19:52Z"/>
    <Property Name="ResourceVersion" Value="465"/>
    <Property Name="Node" Value="f97a7706-2890-4ac7-96ab-6d9405d81d1e"/>
    <Property Name="ServiceAccount" Value="coredns"/>
    <Property Name="Phase" Value="Running"/>
    <Property Name="Started" Value="2021-07-26T17:19:52Z"/>
    <Property Name="Owner" Value="6b3b1428-572e-4cf8-aa6d-0672d5defc04"/>
</Hardware>
```

Field	Description
Name	The name of the pod
Namespace	The namespace in which the pod is encapsulated
UID	Unique identifier for the pod
Created	The time at which the pod was created
Deleted	The time at which the pod was deleted
ResourceVersion	The cluster version in which the pod was last modified

Field	Description
Node	The UID of the node on which the pod is scheduled
ServiceAccount	The service account under which the pod is running
Phase	The runtime state of the pod
Started	The time at which the pod was started
Owner	The UID of the resource that owns the pod
ProductID	The value of the ProductID pod annotation, if set (this is part of the IBM product annotation standard)
ProductName	The value of the ProductName pod annotation, if set (this is part of the IBM product annotation standard)
ProductMetric	The value of the ProductMetric pod annotation, if set (this is part of the IBM product annotation standard)
CloudpakID	The value of the CloudpakID pod annotation, if set (this is part of the IBM product annotation standard)
CloudpakName	The value of the CloudpakName pod annotation, if set (this is part of the IBM product annotation standard)
ProductChargedContainers	The value of the ProductChargedContainers pod annotation, if set (this is part of the IBM product annotation standard)
ProductCloudpakRatio	The value of the ProductCloudpakRatio pod annotation, if set (this is part of the IBM product annotation standard)

MGS_KubernetesContainer

Represents one of the containers in a Pod. You can identify its Pod using the PodUID field.

Note: *To maintain internal consistency within FlexNet Manager Suite with data already collected from Docker, some of the field names differ from those used in the Kubernetes resource.*

Example (notice that lines have been wrapped for presentation):

```
<Hardware Class="MGS_KubernetesContainer"
        Name="kube-system/coredns-558bd4d5db-4w2sc/coredns" Evidence="OS API">
    <Property Name="Name" Value="coredns"/>
    <Property Name="ImageTag" Value="k8s.gcr.io/coredns/coredns:v1.8.0"/>
    <Property Name="PodUID" Value="5c5a521b-7d58-47ca-8983-536bc2ec8c75"/>
    <Property Name="Entrypoint" Value=""/>
    <Property Name="Cmd" Value="-conf /etc/coredns/Corefile"/>
    <Property Name="ImageID"

Value="sha256:cc8fb77bc2a0541949d1d9320a641b82fd392b0d3d8145469ca4709ae769980e"/>
    <Property Name="RestartCount" Value="0"/>
    <Property Name="Status" Value="running"/>
    <Property Name="LastStarted" Value="2021-07-26T17:19:54Z"/>
</Hardware>
```

Field	Description
`Name`	The name of the container
`ImageTag`	The name or `tag` of the image as declared by the user in the specification
`PodUID`	The UID of the pod to which this container belongs
`Entrypoint`	The executable the container should run (not always set, as it may be defined in the container image)
`Cmd`	The arguments to the executable that the container runs (not always set)
`InitContainer`	A Boolean that is true if the container is an `init container` that runs during setup of the `pod`
`CPULimit`	The CPU resource limit, if one is set, in fractional cores
`ImageID`	The unique ID of the container image
`RestartCount`	The number of times the container has been restarted
`Status`	The runtime state of the container

Field	Description
Reason	The reason the container is in the reported state, if present
LastStarted	The time at which the container entered the running state
LastStopped	The time at which the container entered the terminated state
ExitCode	The exit code of the application if the container has terminated

MGS_KubernetesImage

Represents a container image. Example (notice that lines have been wrapped for presentation):

```
<Hardware Class="MGS_KubernetesImage"
        Name="sha256:cc8fb77bc2a0541949d1d9320a641b82fd392b0d3d8145469ca4709ae769980e"
        Evidence="OS API">
    <Property Name="ID"

Value="sha256:cc8fb77bc2a0541949d1d9320a641b82fd392b0d3d8145469ca4709ae769980e"/>
    <Property Name="Names"
        Value="k8s.gcr.io/coredns/coredns@sha256:
        cc8fb77bc2a0541949d1d9320a641b82fd392b0d3d8145469ca4709ae769980e,
        k8s.gcr.io/coredns/coredns:v1.8.0"/>
    <Property Name="SizeBytes" Value="42454755"/>
    <Property Name="ClusterInstalled" Value="2021-07-30T13:03:42.323598116Z"/>
</Hardware>
```

Field	Description
ID	The unique ID of the image
Names	A comma-separated list of names or tags by which the image can be referenced
SizeBytes	The size of the image in bytes
ClusterInstalled	The time at which the image was first observed to be installed on any node in the cluster
ClusterDeleted	The time at which the image was observed to no longer be installed on any node in the cluster

MGS_KubernetesImageInstallation

Represents the installation of a given image on a given node in the cluster. Example (notice that lines have been wrapped for presentation):

```
<Hardware Class="MGS_KubernetesImageInstallation"
        Name="sha256:cc8fb77bc2a0541949d1d9320a641b82fd392b0d3d8145469ca4709ae769980e,
        f97a7706-2890-4ac7-96ab-6d9405d81d1e" Evidence="OS API">
    <Property Name="ImageID"

Value="sha256:cc8fb77bc2a0541949d1d9320a641b82fd392b0d3d8145469ca4709ae769980e"/>
    <Property Name="Node" Value="f97a7706-2890-4ac7-96ab-6d9405d81d1e"/>
    <Property Name="Installed" Value="2021-07-30T13:03:42.323598116Z"/>
</Hardware>
```

Field	Description
`ImageID`	The unique ID of the image
`Node`	The UID of the node on which the image is installed
`Installed`	The time at which the image was first observed to be installed on the node
`Deleted`	The time at which the image was observed to be absent from the node

Inventory from IBM License Service

IBM products (especially Cloud Paks) running in containers in a Kubernetes or OpenShift cluster are required to have their license consumption tracked by the IBM License Service, an application designed for this purpose. It monitors the Pods in the cluster, consuming a set of standardized annotations that must be applied to Pods running IBM products. It uses these annotations to determine what products are in use, and what licensing terms the products use. It exposes this information through a REST API.

Either of the Flexera Kubernetes inventory agent or the lightweight Kubernetes agent can connect with the IBM License Service through its REST API, so that you can import the correct data into FlexNet Manager Suite, and manage your license consumption through a "single pane of glass". The `.ndi` files for uploading inventory extracted from the IBM License Service follow this naming convention: `k8s-ibm-licensing-`*`shortClusterID-timestamp`*`.ndi` (with the *`placeholders`* updated with appropriate values). Example:

```
k8s-ibm-licensing-1656883d-20210726T010000.ndi
```

Note: *The ability to connect with the IBM License Service is not enabled by default in either of the Kubernetes agents. It can be enabled by slightly different processes:*

- *The lightweight Kubernetes agent requires that the `--ibm-licensing` flag is asserted (set to true) (see the example `deployment.yaml` file excerpt below)*
- *The full Flexera Kubernetes inventory agent requires that the `spec.ibmLicensing.enable` attribute of the `KRM` spec must be true:*

```
apiVersion: agents.flexera.com/v1
kind: KRM
...
spec:
  ibmLicensing:
    enable: true
```

Connecting to the REST API for IBM License Service

The two agents need to connect to the REST API endpoint as a URL:

- The lightweight Kubernetes agent uses command-line flags to provide it with the correct URL for the API, and also with the token needed to authenticate with the API (see Options for the Lightweight Kubernetes Agent, with particular attention to `--ibm-licensing` [which must be set to enable the integration], `--ibm-licensing-url`, and `--ibm-licensing-token`). Alternatively, the values may be set in the `deployment.yaml` file:

```
apiVersion: apps/v1
kind: Deployment
...
spec:
  template:
    spec:
      containers:
      - name: agent
        args:
        ...
        - --ibm-licensing
        - --ibm-licensing-url
        - https://ibm-licensing-service-instance.ibm-common-services.svc:8080
        - --ibm-licensing-token
        - VoOMWJijBWuCxSxwgON11w7z
```

- The full Flexera Kubernetes inventory agent has the capacity to discover the API endpoint automatically, or it can be provided with the URL and token for the API in the same manner as the lightweight Kubernetes agent.

Data collection process

Either form of the Kubernetes agent collects *product* and *bundled product* information from the IBM License Service on a daily basis, maintaining a rolling window of 180 days of data. For example, if either agent is deployed on 2022-06-30, the process is as follows:

1. Load the client configuration.
2. Test the availability of the License Service API by issuing a request to its `/version` endpoint.

   a. If the test fails, enter a retry loop with a 5 minute back-off until the test succeeds.

3. Determine the current day (2022-06-30) in the UTC time zone.
4. Determine the end of the period by subtracting one day from the current day (2022-06-29).
5. Determine the start of the period by subtracting 179 days from the end of the period (2021-12-31).
6. Request the product and bundled product data for each day from the start through the end of the period.
7. Write and/or upload an inventory file containing all 180 days of data.
8. Cache the final 7 days of the period.
9. Wait until 01:00:00 the next day (2022-07-01).
10. Request the product and bundled product data for the previous day (2022-06-30).
11. Slide the cache forward one day, storing the new data and deleting the old data.
12. Write and/or upload an inventory file containing the cached 7 days of data.
13. Start a 24-hour timer.
14. When the timer fires request the product and bundled product data for the previous day, slide the cache forward, produce an inventory.

If an agent (more typically, the Flexera Kubernetes inventory agent) is permitted to write to disk, it persists the cached data, which, if the agent is restarted, allows it to restore data from the cache, reducing the load on the IBM License Service. So for a restart, if there is data already in the cache, the agent validates whether the data covers the entire desired period:

- If so, it resumes the above process from step 9
- If not (for example, if the agent was disabled for several days), it starts from step 6 in the above process and requests data for the missing days.

The uploaded inventory format

Within the normal `.ndi` file format, the IBM License Service data is encapsulated in a `ServiceProvider` block. This block includes:

- The `/version` and `/health` data (collected from the matching endpoints of the API)
- The date range that the data covers
- A series of zero or more `Product` and `BundledProduct` elements showing the related data recovered that day. These elements have added `date` and `clusterid` attributes, and otherwise mirror the IBM License Service data model (for details, see https://www.ibm.com/docs/en/cpfs?topic=service-license-swagger-api-schema).

The following example shows the structure of the `ServiceProvider` block within the `.ndi` file. The lines have been wrapped here for presentation, and the example uses dummy data:

```
<ServiceProvider Type="IBM License Service" LastInventoryResult="0"
        LastInventoryError="" Name="d8259158-fdfe-4ba3-ae28-be62a51df7f9">
    <Property Name="Version" Value="1.6.0"/>
    <Property Name="BuildDate" Value="Mon Jul 12 16:45:52 UTC 2021"/>
```

```
    <Property Name="IncompleteAnnotationCount" Value="0"/>
    <Property Name="IncompleteAnnotationPods" Value=""/>
    <Property Name="StartDate" Value="2021-07-24"/>
    <Property Name="EndDate" Value="2021-07-25"/>
    <Product date="2021-07-24T00:00:00Z" id="eb9998dcc5d24e3eb5b6fb488f750fe2"
        name="IBM Cloud Pak for Data" metricName="VIRTUAL_PROCESSOR_CORE"
        metricQuantity="2" clusterid="d8259158-fdfe-4ba3-ae28-be62a51df7f9"/>
    <BundledProduct date="2021-07-24T00:00:00Z"
        cloudpakId="eb9998dcc5d24e3eb5b6fb488f750fe2"
        cloudpakName="IBM Cloud Pak for Data" cloudpakVersion="3.2.1"
        productId="fb1b19783983ec76198729a9b945723"
        productName="IBM Cloud Pak for Data Control Plane"
        metricName="VIRTUAL_PROCESSOR_CORE"
        cloudpakMetricName="VIRTUAL_PROCESSOR_CORE" metricConversion="1:1"
        metricConvertedQuantity="2" metricMeasuredQuantity="2"
        clusterid="d8259158-fdfe-4ba3-ae28-be62a51df7f9"/>
    <Product date="2021-07-25T00:00:00Z" id="eb9998dcc5d24e3eb5b6fb488f750fe2"
        name="IBM Cloud Pak for Data" metricName="VIRTUAL_PROCESSOR_CORE"
        metricQuantity="2" clusterid="d8259158-fdfe-4ba3-ae28-be62a51df7f9"/>
    <BundledProduct date="2021-07-25T00:00:00Z"
        cloudpakId="eb9998dcc5d24e3eb5b6fb488f750fe2"
        cloudpakName="IBM Cloud Pak for Data" cloudpakVersion="3.2.1"
        productId="fb1b19783983ec76198729a9b945723"
        productName="IBM Cloud Pak for Data Control Plane"
        metricName="VIRTUAL_PROCESSOR_CORE"
        cloudpakMetricName="VIRTUAL_PROCESSOR_CORE" metricConversion="1:1"
        metricConvertedQuantity="2" metricMeasuredQuantity="2"
        clusterid="d8259158-fdfe-4ba3-ae28-be62a51df7f9"/>
</ServiceProvider>
```

Field	Description
`BuildDate`	The date when the current version of the IBM License Service was compiled for release.
`IncompleteAnnotationCount`	The number of pods found with incomplete annotations about IBM Cloud Pak applications in use within the pod.
`Version`	The full version the IBM License Service
`IncompleteAnnotationPods`	A list of the pods where the required annotations are incomplete.

Field	Description
`StartDate`	If this attribute is empty, the API returned products and bundled products for the last 30 days as a default. When either `StartDate` or `EndDate` is specified in the query to the API, both must be supplied. The `StartDate` is the first day for which details are returned (that is, this value is inclusive).
`EndDate`	See `StartDate` for more details. The `EndDate` is the day after the last data was collected (that is, this value is exclusive). Therefore in the example above, data for just one day (2021-07-24) is returned.
`Product`	An element with the following attributes: • `date` — The date when this particular product was found in the cluster • `id` — The internal IBM ID for this application • `name` — The published name of the application • `metricName` — The license metric associated with this product (metrics may be used to filter the applications returned from the IBM License Service API) • `metricQuantity` — The value associated with the metric (for example, if the `metricName` is `VIRTUAL_PROCESSOR_CORE`, a quantity of 2 means that the product in this row consumes 2 cores in the current cluster) • `clusterid` — The identifier for the cluster where this product was found installed.

Field	Description
`BundledProduct`	An element with the following attributes: • `date` — The date when this bundled product was found in the cluster • `cloudpakId` — The internal IBM ID for the Cloud Pak containing this `BundledProduct` • `cloudpakMetricName` — The metric associated with this bundle (metrics may be used to filter the data returned from the IBM License Service API) • `cloudpakName` — The published name of the bundle • `cloudpakVersion` — The version of the installed bundle • `productId` — The internal IBM ID for this product within the Cloud Pak bundle • `productName` — The name of this application within the bundle • `metricName` — The metric associated with this product within the bundle • `metricConversion` — a string representing the ratio to be applied when converting the `metricMeasuredQuantity` to the `metricConvertedQuantity` • `metricMeasuredQuantity` — The value for the metric that the IBM License Service recovered for the installation • `metricConvertedQuantity` — The result of applying the `metricConversion` ration to the `metricMeasuredQuantity` • `clusterid` — The identifier for the cluster where this product was found installed.

III

Collecting Inventory from Container Images

This part covers a tool for making a "static analysis" of software inventory running in containers, without having any impact on containers that are running in production.

The first chapter outlines the approach to inventory collection, the prerequisites for running the tool, and the way to obtain it.

The second chapter goes into depth on how the tool works, with special considerations for handling the inventory tool and the container image(s) you want to investigate; and covers all the options that may be used with this tool.

1

Container Image Inventory Tool imgtrack

To track and correctly license the software deployed within a container, you must know either:

- What software is available within the container while the container is running; or
- What software is present in the *image* (the application binaries, libraries, configuration files, language run-times, and so on) from which the container is (or is going to be) instantiated.

The first approach is straight-forward: you can collect software inventory from a running container using (for example) the Flexera Kubernetes inventory agent. This can use the "zero footprint inventory collection" method, where the FlexNet Inventory Scanner (on UNIX-like platforms, `ndtrack.sh`) is injected into a running container, executed to collect software inventory, and then removed. This strategy is very convenient, but it can be intrusive and may require permissions within the container management platform that are not acceptable for high security organizations. For this reason, Flexera Kubernetes inventory agent allows the feature to be disabled; and the alternative lightweight Kubernetes agent does not include the inventory-collection feature at all. In these cases, some other means of obtaining an inventory of the software within container images must be used.

The `imgtrack` tool uses the second approach. Rather than operating on live application containers as they are executing, `imgtrack` is run separately from the container management platform, ideally as part of a continuous integration/continuous deployment (CI/CD) system. This is rather like the program "static analysis" technique where the text of the code is analysed in detail before it is run, so we refer to this strategy as static analysis of the container image.

`imgtrack` is a Bash shell script invoked on the command line of a suitable Linux-based computer. It leverages the standard FlexNet Inventory Scanner and a locally-running instance of Docker to produce a standard Flexera inventory (`.ndi`) file from a target container image, optionally uploading that result to your chosen inventory beacon.

In this chapter:

- The prerequisites for the Linux device(s) where you may wish to execute `imgtrack`
- How to obtain and position the `imgtrack` script.

Prerequisites for imgtrack

The `imgtrack` script must be executed on a device that meets all of the following requirements:

- Uses a supported hardware architecture: x86_64
- Runs a supported version of the Linux operation system (see supported versions for FlexNet inventory agent in)
- Runs the `bash` shell
- Provides standard Linux utilities: `awk`, `basename`, `cat`, `cp`, `cut`, `grep`, `head`, `mktemp`, `tail`, `tar`, `tr`, `uname`
- Has a running instance of Docker

Note: *Other container runtimes (such as podman) are not currently supported. Docker commands are issued directly by `imgtrack`, so the user account used to invoke `imgtrack` must have sufficient privileges to run the commands. See the Docker documentation for details on configuring user access.*

- Has network access to any relevant image registries where target images are saved
- Has the target container images present in the image index for the local Docker instance, meaning that one of the following must apply:
  - An image may be pulled prior to invoking `imgtrack`
  - The image may be built on the local computer where `imgtrack` is executed
  - `imgtrack` may be invoked with the `--pull` option

Remember: *If you are using the `--pull` option, you must use the `docker login` command to authenticate with any private registries before invoking `imgtrack`.*

- Has network connectivity to access the inventory beacon where `ndtrack` is to upload the collected inventory

Tip: *It is possible (if somewhat less convenient) to use the `--output-dir` option to save the inventory `.ndi` file to a folder on the host device, and then manually move the file to another location for upload; but it is simpler if the container running the derived image has network access to the specified inventory beacon.*

- The target image must include one of the following supported C standard library implementations:
  - `glibc` – the standard GNU implementation
  - `muslc` – the standard in the Alpine Linux distribution.

Tip: *If the target image does not include any C standard library, or if it includes a different C library that is neither of the above, `imgtrack` considers the image to be unsupported and exits with an error.*

It is assumed in these requirements that `imgtrack` and the Docker daemon are running on the same device.

Downloading the imgtrack Script

The `imgtrack` script is available in the common tar archive that includes both Flexera Kubernetes inventory agent and the lightweight Kubernetes agent.

To download the `imgtrack` script:

1. Log on to a device that:
   - Runs a supported version of Linux
   - Has a web browser with network access to the web application server for FlexNet Manager Suite, where your account must have operator privileges to see the appropriate page
   - Has a running instance of Docker
   - Either hosts an OCI container registry, or has network access to an OCI container registry, that is available to your Kubernetes cluster.

   ***Tip:** The `imgtrack` script does not have any ability to log into your container registry. This process is simplified, then, if you use an account that has login privileges for the image registry containing the target image(s) from which you want to collect software inventory. Remember to use the*

   ```
   docker login
   ```

   command prior to executing the `imgtrack` script.

2. In your web browser, navigate in FlexNet Manager Suite to **Discovery & Inventory > Settings**.

   The **Inventory Settings** page displays.

3. Expand the **Inventory agent for download** section.

4. Click **Download Flexera Kubernetes inventory agent** and save the downloaded archive file to a suitable location.

   ***Tip:** Despite the name on the link, the downloaded archive also includes the `imgtrack` script.*

5. Extract the folders from the downloaded archive, for example with the following command line:

   ```
   tar xzf flexera-krm-operator.tar.gz
   ```

6. Move into the directory that was extracted from the archive:

   Replace the placeholder *`x.y.z`* with the version number of the first extracted folder, and replace *`a.b.c`* with the version number included for the `imgtrack` folder.

   ***Tip:** These version numbers may not be the same. It may be easier to do this in two steps, where you can check the version number in the folder name.*

   ```
   cd flexera-krm-operator-x.y.z
   ```

```
cd imgtrack-a.b.c
```

This last folder contains the complete `imgtrack` script and associated files.

Because `imgtrack` is a fully self-contained script, it can be copied to, and run in, any path of your choosing, such as `/usr/bin` or `/opt/managesoft/bin` or another path to suit your CI/CD processes.

2

How imgtrack Works

`imgtrack` is a Bash shell script invoked on the command line of any Linux computer meeting its requirements. It is a utility whose job is to arrange for the execution of the FlexNet Inventory Scanner in a context where it can access the file system content of a container image, without interfering with any container running in a production environment.

To do this, `imgtrack` derives an image (the *derived image*) from the *source image* that is the target for inventory collection. The derived image is created in the standard Docker way, by adding a layer to a copy of the source image, where the extra layer in this case contains the FlexNet Inventory Scanner and related files. The entry point for the derived image (that is, the process invoked when a container is instantiated from the image and run) is to run the FlexNet Inventory Scanner.

Both the derived image and any container run from it are short-lived – both are destroyed when the inventory process is completed. Of course, this does not affect the source image in any way.

An operator (or perhaps a part of your CI/CD automation) invokes the script with a target image to inventory, and your preferred options (for which see Options for the imgtrack Script):

```
imgtrack image [options]
```

The script then performs the following operations:

1. **Optionally, pull the source image:** If the `--pull` option was specified, the target image can be pulled from the registry. This requires that the credentials used to run `imgtrack` have read permissions from the registry, and that the operator has logged into the repository (if it required authentication) before invoking `imgtrack` (since, to avoid setting authentication parameters on the command line, `imgtrack` does not support logging in to any registry).
2. **Load image metadata:** `imgtrack` uses the `docker inspect` command to collect metadata including both the image ID and the Repo Digest from Docker (for details, see Identifying Container Images). This also verifies that the source image exists in the local Docker image index – if this command fails, it is a fatal error for this run of `imgtrack`.
3. **Locate ndtrack source:** The `--from-ndtrack` option may be used to specify a custom installation of the FlexNet Inventory Scanner (`ndtrack.sh`) already existing on the local Linux device (or less commonly, perhaps, the `--local-ndtrack` option may point to use of the installation in the default location on this device). These options, and the related installations, are not mandatory, since the `imgtrack` script includes a tarball of `ndtrack.sh` with its platform-related versions of the `ndtrack` inventory component.

4. **Determine ndtrack platform:** The `imgtrack` script uses the `uname` utility on the host (the local Linux device where the script is running) to determine which platform-specific version of `ndtrack` must be run. Then `imgtrack` runs a container from the source image where it uses the `ldd` command to determine which implementation (if any) of the C standard library is available.

   Tip: *No other software is run in this container, and it is removed immediately after the check for the C library. You may, instead, bypass this check on the C library implementation using the `--libc-variant`option to specify the implementation that is available.*

5. **Create working directory:** `imgtrack` requires several temporary files during operation, and uses the `mktemp` utility to create a work directory (and subdirectories) to hold these. The work directory (and subdirectories, and contents) are by default deleted before `imgtrack` exits (even with an error), although you may prevent that clean-up with the `--no-cleanup-files` option.

6. **Extract ndtrack into working directory:** Using the appropriate tarball selected at step 3, `imgtrack` installs the platform-specific version of `ndtrack` ready to collect software inventory.

7. **Search for InventorySettings.xml:** The `InventorySettings.xml`, as updated from time to time with the downloads of the Application Recognition Library, extends the inventory-gathering functionality of `ndtrack` especially in areas like Oracle and Microsoft inventory. The script looks for this valuable file in the default installation folder (on this Linux device), or in the path specified with the `--inventorysettings-path` option.

8. **Construct Dockerfile:** This manifest instructs Docker on how to build an image. This takes the source image as a base, adding a layer for `ndtrack` and `InventorySettings.xml`, and configures the command line for `ndtrack`.

   Tip: *Because `ndtrack` requires that it runs as the `root` user, `imgtrack` explicitly sets to user to `root` in the Dockerfile.*

   To inspect the Dockerfile, run `imgtrack` with the `--no-cleanup-files` option. Although the file name (created with `mktemp`) is unpredictable, the file is contained within the work directory.

9. **Build derived image:** `imgtrack` now uses the `docker build` command to build the derived image. Several labels are applied to the image at build time (see Labels for the Derived Image for details). To review the derived image without instantiating a container, use the `--build-only` option, which causes `imdtrack` to exit at this point *without* deleting the derived image.

10. **Run container from derived image:** `imgtrack` now uses the `docker run` command to instantiate and run a container based on the new derived image. The container executes the `ndtrack` component, which collects software inventory from inside the container, saving the results in an `.ndi` file. Normal practice is to specify the `--beacon` option, so that the tracker can upload the `.ndi` file as soon as it is ready to the inventory beacon at that URL.

    Tip: *This requires that the container must be attached to a network that can access the inventory beacon. Use the `--network` option to specify a suitable Docker network to which the container is attached.*

    If the inventory beacon serves over HTTPS, the CA certificate bundle needed to verify the inventory beacon's certificate must be available in the container. If the source image provides the needed certificate bundle, no further action is needed. If the source image does not supply an appropriate certificate bundle, it can be injected into the derived image using the `--ca-certificates` option.

11. **Copy inventory into host directory:** If the `--output-dir` option was set to a directory on the host system, `imgtrack` copies the saved `.ndi` file into the final directory on the host given in the option's value. This option may be used in addition to the beacon upload option, but at least one of the two should be used.
12. **Delete derived image:** By default, `imgtrack` arranges for clean-up after the container terminates by using the `--rm` option to the `docker run` command. This removes both the derived image and the work directory (along with all the files in it, of course). To retain artifacts for inspection, troubleshooting, or evaluation, see the various `--no-cleanup-*` options – after which the preserved artifacts need to be deleted manually.

Handling the ndtrack Binary

The purpose of `imgtrack`, given a target image, is to create a derived image, temporarily run a container from it, and deliver to it the inventory component (`ndtrack`) of the standard FlexNet Inventory Scanner.

Choosing the ndtrack binary

The `imgtrack` tool needs access to a copy of `ndtrack` appropriate to the particular Linux platform. Like the FlexNet Inventory Scanner (`ndtrack.sh`) for Linux platforms, `imgtrack` includes (concatenated on the end of the script) a tarball of various versions of `ndtrack` to extract and run on the relevant Linux platform(s). It determines the platform using the `uname` utility, and installs and runs the appropriate edition of `ndtrack` for the platform. It is normal, and best practice, to simply allow `imgtrack` to extract the appropriate edition of `ndtrack` from its embedded tarball. However, where this does not suit your corporate strategies, there are two alternatives:

- You can instruct `imgtrack` to use a pre-installed copy of `ndtrack.sh` (this may have been installed through adoption or through third-party deployment). For success, this must be a standard installation, where the tracker is located in `/opt/managesoft/libexec/support`. To look here for the tracker, and if found run that version instead of choosing one from the attached tarball, use the `--local-ndtrack` option when invoking `imgtrack`.
- Otherwise, you can direct `imgtrack` to use a copy of the FlexNet Inventory Scanner (`ndtrack.sh`) saved in a custom location, using the `--from-ndtrack` option. (If both these options are specified, this `--from-ndtrack` option takes precedence.)

Compatibility is only guaranteed between `imgtrack` and the versions of `ndtrack` embedded within its own tarball. If you choose either of the above options, it is your responsibility to ensure both the compatibility and the integrity of your copy of `ndtrack.sh`.

Providing the appropriate libc

The `ndtrack` binary is implemented in the C/C++ code family, and requires the standard runtime and libraries to execute. However, the target container image may be constructed without a C language runtime, standard system libraries, or typical system tools, directory layouts, or configurations; or it may contain an incompatible library. When `imgtrack` creates a derived image, any C library included in the source image is, of course, also included in the derived image.

To determine what C library (if any) is included in the target (source) image, `imgtrack` briefly instantiates a container from the source image.

Tip: *Under no circumstances does `imgtrack` run any of the software within this temporary container. It invokes the*

command

```
ldd --version 2>&1
```

to capture output that includes the specific implementation of the C library (if any). Immediately afterwards, the container is deleted.

If the command fails, or if the output from the command does not contain the information to identify the C library implementation, `imgtrack` exits with an error, as the image is unsupported.

You can bypass the process for detecting the standard C library implementation in the source image by directly specifying the implementation to use with the `--libc-variant` option, which must identify one of the supported implementations:

- `glibc` – the standard GNU implementation
- `muslc` – the standard in the Alpine Linux distribution.

Example:

```
imgtrack example:latest --libc-variant glibc
```

Identifying Container Images

There are multiple ways to identify an image used to instantiate containers.

Image name

A container image has a name which may contain *lower-case* letters, numbers, and separators (defined as a period, one or two underscores, or one or more dashes – none of which may be the first or last character of a component of the image name). The name may be made up of multiple components:

- The *repository* component, which may include slash-separated elements:
  - Optionally, the hostname of the registry storing the image. In additional to normal DNS rules for host naming, the registry hostname may not contain underscores.
  - Optionally, a port number in the normal format `:8080`.

    ***Tip:** If the registry hostname is not present, the image is assumed to be saved in Docker's public registry, located at `registry-1.docker.io` by default.*
  - The name of the registry stored on that host
- A colon (`:`) separator
- A tag name of up to 128 characters, using valid ASCII upper- and lower-case letters, digits, underscores and (except as the first character of the tag) periods and dashes. Each tag:
  - Refers to exactly one discrete image at a given point in time
  - May be one of several tags that point to a single image

- May be moved to refer to a different image.

Image ID

The images used to instantiate containers are widely referenced by using names; but within the container runtime, the images are identified using IDs. This is because, unlike names (which can be transferred between images), the ID is a digest generated as a hash of the image content. For this reason, an ID always uniquely identifies the image in its current form. Any change in the image, whether in its component layers or in its additional metadata, results in a change in its hash digest. So the image ID always, and uniquely, identifies the one image in its current state; and it is neither possible for this ID to point to a different image, nor for a given image (in an unchanged state) to have a different ID.

In Docker you can inspect the image ID for a named image like this:

```
$ docker inspect images.example.com/foo:latest --format={{.ID}}
sha256:b2fcd079c1d403dc1dba5397ca1bca606f17ebcf99b03b66c59941929acff57c
```

Repo Digest

Container images are stored and shared through the *registry*, with the relationship between image and registry stored in some additional metadata. This metadata gives rise to the *Repo Digest*, which includes both the registry component of the image name (the portion before the colon `:`), and its tag. Although technically the Repo Digest is an array, the `imgtrack` script always uses the first value in that array and ignores any others than may be present.

Because the Repo Digest tracks the relationship between image and repository, it is only generated when the image is first pushed to the registry. An image built on the local computer and not yet pushed to a registry does not have a Repo Digest. It is important, therefore, to push any local image to the repository *before* invoking the `imgtrack` script.

In Docker you can view the Repo Digest(s) for an image like this:

```
$ docker inspect images.example.com/foo:latest --format='{{join .RepoDigests ","}}'
images.example.com/
foo@sha256:6647385dd9ae11aa2216bf55c54d126b0a85637b3cf4039ef24e3234113588e3
```

Choosing the ID to use

It's helpful to keep the difference between the true image ID and its Repo Digest clearly in mind, for these reasons:

- The Kubernetes API (mis-)uses the label `ImageID`, but the content in this value is actually the image's Repo Digest.
- When either the Flexera Kubernetes inventory agent or the lightweight Kubernetes agent reports inventory from the Kubernetes API, it therefore reports the image's Repo Digest.
- When `imgtrack` inspects the source image, it attempts to capture both the image ID and the Repo Digest.

  Remember: *The Repo Digest for a locally-built image only exists after the image has been pushed to the repository. Be sure that each image has been pushed to a repository (or is a shared image that has previously been pushed, and has now been pulled from the repository for inspection) before invoking the `imgtrack` script.*

- Correct merging of records from the Kubernetes agent(s) and the `imgtrack` script requires that they are using the same ID for each image.

- Therefore, if the Repo Digest is not empty, `imgtrack` uses its first entry to identify the image, allowing for simple matching with the ID values collected from Kubernetes. If the Repo Digest is empty/missing, `imgtrack` uses the image ID.

Labels for the Derived Image

The derived images created by `imgtrack` are not tagged with a name. If you have used one of the options that prevent cleanup of the derived image(s), you may want to manage the images or eventually delete them. To facilitate this, the derived images are labelled with a variety of information.

Firstly, all the derived images have a label `app.flexera.com` with the value `imgtrack`. The `--filter` option to the `docker image ls` command can be used to list all images tagged with this label:

```
docker image ls --filter=label=app.flexera.com=imgtrack
```

In the output from this command, the `REPOSITORY` and `TAG` field have the value `<none>`, and cannot be used for image management. The `IMAGE ID` field, however, has the short-form ID (the first 12 characters of the ID) of the image, which can be used in subsequent `docker` commands to interact with the image.

Other labels:

- Describe the source image
- Identify the versions used of `imgtrack` and `ndtrack`
- Link the image to the `.ndi` inventory file that was (or would have been) generated by `ndtrack` for the image.

You can extract the labels from the image metadata in any of the following ways:

- Run the `docker inspect` command, and visually identify the labels in the JSON data the command produces.
- Use Docker's templating feature to extract and format the labels (lines wrapped for presentation here):

```
shell$ docker inspect b069f4a6dd3b
    --format='{{range $k, $v := .Config.Labels}}{{printf "%s: %s\n" $k $v}}{{end}}'
app.flexera.com: imgtrack
imgtrack.flexera.com/filename: imgtrack-sha256_6647385dd9ae.ndi
imgtrack.flexera.com/id:
    sha256:6647385dd9ae11aa2216bf55c54d126b0a85637b3cf4039ef24e3234113588e3
imgtrack.flexera.com/managesoft_version: 17.3.0
imgtrack.flexera.com/version: 1.0.0
source.imgtrack.flexera.com/image_id:
    sha256:b2fcd079c1d403dc1dba5397ca1bca606f17ebcf99b03b66c59941929acff57c
source.imgtrack.flexera.com/repo_digest:
    postgres@sha256:6647385dd9ae11aa2216bf55c54d126b0a85637b3cf4039ef24e3234113588e3
source.imgtrack.flexera.com/tag: postgres:13
```

- Use the `jq` tool to extract the labels from the JSON data (lines wrapped for presentation):

```
shell$ docker inspect b069f4a6dd3b | jq .[0].Config.Labels
{
```

```
  "app.flexera.com": "imgtrack",
  "imgtrack.flexera.com/filename": "imgtrack-sha256_6647385dd9ae.ndi",
  "imgtrack.flexera.com/id":
     "sha256:6647385dd9ae11aa2216bf55c54d126b0a85637b3cf4039ef24e3234113588e3",
  "imgtrack.flexera.com/managesoft_version": "17.3.0",
  "imgtrack.flexera.com/version": "1.0.0",
  "source.imgtrack.flexera.com/image_id":
     "sha256:b2fcd079c1d403dc1dba5397ca1bca606f17ebcf99b03b66c59941929acff57c",
  "source.imgtrack.flexera.com/repo_digest":

"postgres@sha256:6647385dd9ae11aa2216bf55c54d126b0a85637b3cf4039ef24e3234113588e3",
  "source.imgtrack.flexera.com/tag": "postgres:13"
}
```

As an exercise, the following shell script snippet illustrates how you could find and display the labels for every derived image built by `imgtrack` on the Linux device:

```
for image in $(docker image ls --filter=label=app.flexera.com=imgtrack
--format={{.ID}}); do
  echo "$image"
  docker inspect $image
      --format='{{range $k, $v := .Config.Labels}}{{printf "\t%s: %s\n" $k $v}}{{end}}'
done
```

Options for the imgtrack Script

Invocation

`imgtrack` is a shell script run from the command line of a Linux-based computer that has a local instance of Docker running, and, if necessary, is authenticated with the remote registries that store the container images used by your enterprise. The basic syntax identifies the container image under investigation (either by name or by ID), followed by zero or more option arguments.

```
imgtrack image [options]
```

If the image name is used, it follows the Docker convention of the repository/image name (shown in this page as *`example`*), a colon as a separator, and a current tag for this image within that repository (shown in this page as *`latest`*).

```
imgtrack example:latest --beacon https://mybeacn.example.com:443/leaveK8s
```

An alternative mode is to provide the `--help` option to list details about the command line:

```
imgtrack --help
```

Option values

Options follow the standard UNIX conventions.

- Short-form options have a single dash followed by a single letter, and sometimes a space and a value
- Long-form options have two dashes, the option name, and sometimes a space and a value
- Some options are Boolean, and take effect (become true) solely from their inclusion (they do not need a value).

Placeholders for values to supply are shown *thus*.

Available options (listed in alphabetical order) include:

`--beacon` *URL*
`-b` *URL*

Type: Valid URL

Default: Unset

The URL of an inventory beacon to which collected inventories are uploaded immediately after collection. Must include the protocol (either `http://` or `https://`), the host name, an optional port number (if used, separated from the host name by a colon), and any necessary path elements to reliably reach the inventory beacon. The value may include any or none of the `/ManageSoftRL/Inventory` path components used within the inventory beacon — these will be automatically appended by `imgtrack` if omitted from the flag (as in the example below).

Tip: *Uptime on the inventory beacon, and network reliability on the path from the Linux device to the inventory beacon, are critical to inventory gathering. Because the derived image container is removed after the inventory gathering exercise is completed, there is no nightly catch-up of inventory uploads to recover from temporary network outages. The inventory beacon specified with this flag is the only one used by FlexNet Inventory Scanner when invoked by `imgtrack`. Also note that if you are using the `https` protocol, `imgtrack` supports only standard TLS to authenticate those communications.*

If you do not wish to upload inventory to any inventory beacon, or if you want to provide a local file backup to recover from possible network interruptions, see `--output-dir`.

Example:

```
--beacon https://mybeacn.example.com:443/cntnrs
```

`--build-only`	*Type:* Boolean *Default:* Unset Stop after building the derived image. The container is not run and no inventory is produced. The ID of the derived image, and the docker command that would have been run, are printed to the screen, and then `imgtrack` exits. This option also automatically sets `--no-cleanup-image`. *Example:* `imgtrack example:latest --build-only`

`--ca-certificates path`	*Type:* String (valid path on the host computer to certificate bundle or folder)
	Default: Unset
	Copy the Certificate Authority (CA) certificates at the given path on the host computer into the derived image, and configure `ndtrack` to use them. The path may refer to:
	• A PEM-encoded file consisting of a bundle of CA certificates (in this case, the SSLCACertificateFile option for `ndtrack` is set)
	• A directory containing a number of PEM-encoded CA certificates (in this case, the SSLCACertificatePath option for `ndtrack` is set).
	In either case, the file or directory is copied into the work directory, and then into the derived image.
	Example:
	`imgtrack example:latest --ca-certificates /etc/ssl/certs/ca-bundle.crt`
	`imgtrack example:latest --ca-certificates /etc/ssl/certs`

`--cpus float`	*Type:* A floating-point number *Default:* Unset (which means use all CPUs, which is equivalent to running a process outside a container, directly on the host, without a CPU limit) Specifies the number of CPUs as the upper limit to assign when running the test container from the derived image. The value is passed directly to the `docker run` command (see the Docker documentation for details). *Example:* `imgtrack example:latest --cpus 0.5`

`--from-ndtrack path`	*Type:* Valid file path and file name for an installed version of the FlexNet Inventory Scanner self-installing script (`ndtrack.sh`) on the local device *Default:* Unset For inventory gathering, use the appropriate `ndtrack` binary from the `ndtrack.sh` self-installing script at the given path. This option can be useful in cases where the version of `ndtrack` embedded in `imgtrack` is not your organization's accepted version, but your approved version of `ndtrack.sh` has already been deployed to a custom location on the Linux device. Notice that this is the dominant setting for `ndtrack.sh`. If the `--local-ndtrack` option is also given, it is ignored in favor of this option. Compatibility is only guaranteed between `imgtrack` and the version of `ndtrack.sh` that it embeds within itself. If you specify `--from-ndtrack`, it is possible that the installed version of `ndtrack` used will not support all of the features necessary for proper operation of `imgtrack`. It is also your responsibility to ensure the integrity of, and trust in, the copy of `ndtrack.sh` that is used. *Example:* `imgtrack example:latest --from-ndtrack /path/to/ndtrack.sh`
`--inventorysettings-path path`	*Type:* String (valid path to an installed copy of `InventorySettings.xml`) *Default:* Unset (meaning to look for the `InventorySettings.xml` file locally installed in the default path, `/var/opt/managesoft/tracker/inventorysettings/InventorySettings.xml`.) To extend the functionality of the inventory component, look for (and if found, use) the copy of `InventorySettings.xml` found in the path provided. (If you need a copy of this file, which extends the functionality of `ndtrack` especially for Oracle and Microsoft inventory gathering, it is available on any inventory beacon in the default path `%ProgramFiles%\Flexera Software\Inventory Beacon\RemoteExecution\Public\Inventory`, as defined in the Windows share `mgsRET$`.) ***Tip:*** *If `--no-inventorysettings` is present, this option is ignored.* *Example:* `imgtrack example:latest --inventorysettings-path /some/directory`

`--libc-variant name`

Type: A string value that is exactly one of:

- `glibc`
- `muslc`

Default: Unset

Use the named C library implementation, assumed now to be provided by the image under investigation. Skip the test to determine which C library implementation to use.

By default, `imgtrack` executes a container from the source image, running the command

```
ldd --version 2>&1
```

to determine which C library implementation to use based on the output (or else determine that the image is not supported). If `--libc-variant` is given, this test is skipped.

Example:

```
imgtrack example:latest --libc-variant muslc
```

`--local-ndtrack`

Type: Boolean

Default: Unset (uses the `ndtrack.sh` tarball bundled with `imgtrack`)

For inventory gathering, use the appropriate `ndtrack` binary from the `ndtrack.sh` "zero-footprint" self-installing script already present in the default location on the local computer at `/opt/managesoft/libexec/support/ndtrack.sh`. This option can be useful in cases where the version of `ndtrack` embedded in `imgtrack` is not your organization's accepted version, but your approved version of `ndtrack.sh` has already been deployed to the Linux device.

Tip: *If the `--from-ndtrack` option is given, it takes precedence, and this option is ignored.*

Compatibility is only guaranteed between `imgtrack` and the version of `ndtrack.sh` that it embeds within itself. If you specify `--local-ndtrack`, it is possible that the installed version of `ndtrack` used will not support all of the features necessary for proper operation of `imgtrack`. It is also your responsibility to ensure the integrity of, and trust in, the copy of `ndtrack.sh` that is used.

Example:

```
imgtrack example:latest --local-ndtrack
```

`--memory size`

Type: A special combination of an integer, followed immediately by a single character indicating the unit – one of b (bytes), k (kilobytes), m (megabytes), or g (gigabytes)

Default: Unset

Set a memory limit when running the container. This value is passed through directly to the `docker run` command, where the minimum value is `4m` (see the Docker documentation for details).

Example:

```
imgtrack example:latest --memory 256m
```

`--ndtrack-log`

Type: Boolean

Default: Unset

Collect the log file written by `ndtrack` and print it to standard output. This is mostly useful for troubleshooting issues in the operation of `ndtrack` itself, such as issues uploading to the inventory beacon or issues with specific `ndtrack` features.

The `ndtrack` component does not support logging directly to standard output. To collect the logs, `imgtrack` creates a temporary directory within the working directory. The directory is mounted into the container at runtime, and the log file is written into it.

Tip: *This is the same directory as when `--output-dir` is used.*

Once the container exits, `imgtrack` writes the contents of the log file to the screen.

Example:

```
imgtrack example:latest --ndtrack-log
```

`--ndtrack-opt option`
`-o option`

Type: String

Default: Unset

Supply an option directly to the inventory component of the FlexNet Inventory Scanner (`ndtrack`). This should be needed only in rare circumstances. For more information about options for FlexNet Inventory Scanner, see Preferences.

Tip: *Some `ndtrack` options are crucial to the operation of `imgtrack`, and cannot be overwritten. These options are visible within the `imgtrack` script.*

Example:

```
imgtrack example:latest -o LowProfile=True
```

`--network name`

Type: String

Default: Unset

Attach the container to the named network. This option may be useful in cases where the default Docker network is not able to communicate with the inventory beacon, but a different Docker network can do so. The value is passed directly to the `docker run` command (see the Docker documentation for details).

Example:

```
imgtrack example:latest --network foo
```

`--no-cleanup-all`

Type: Boolean

Default: Unset

An alias for `--no-cleanup-files --no-cleanup-image --no-cleanup-container`.

Example:

```
imgtrack example:latest --no-cleanup-all
```

`--no-cleanup-container`

Type: Boolean

Default: Unset (meaning that `imgtrack` adds the `--rm` option to the Docker command, so that the container is deleted as soon as it exits)

Do not delete the container after it has exited (by omitting the `--rm` option from the Docker command). Eventually, this container needs to be deleted manually.

> ***Remember:*** *The image on which the container is based cannot be deleted while the container exists. For this reason, if you set `--no-cleanup-container`, it automatically also sets `--no-cleanup-image`, so that in due course, the derived image also needs to be deleted manually.*

This is only useful for niche troubleshooting situations where you need to access the content of the container after the operation has completed.

Example:

```
imgtrack example:latest --no-cleanup-container
```

`--no-cleanup-files`

Type: Boolean

Default: Unset

Do not delete the working directory or any of the files contained within it. Write a message to standard output with the working directory's path (because the directory was created using the *mktemp* utility, meaning that the directory is unpredictably named). This is only useful for troubleshooting or evaluation.

Example:

```
imgtrack example:latest --no-cleanup-files
```

To conduct a dry run for evaluation purposes, combine this with the `--build-only` flag:

```
imgtrack example:latest --no-cleanup-files --build-only
```

`--no-cleanup-image`	*Type:* Boolean *Default:* Unset (meaning to delete the derived image before `imgtrack` exits, regardless of success or failure) Do not delete the derived image. ***Tip:*** *The derived image is not tagged, so it does not have a name. In the normal output from the `docker image` command, such images appear with a name and tag of "<none>". (Despite being unnamed, the derived images are labeled with information that can be used to manage them.) Note that the `docker image prune` command deletes these images.* *Example:* `imgtrack example:latest --no-cleanup-image`
`--no-file-evidence`	*Type:* Boolean *Default:* Unset (false) By default, `imgtrack` enables a set of options to `ndtrack` that enable gathering file evidence through SWID tags. While the need to do so will be rare, this flag can be used to disable these options, so that file evidence from SWID tags is *not* included in the inventory. *Example:* `imgtrack example:latest --no-file-evidence`
`--no-inventorysettings`	*Type:* Boolean *Default:* Unset (meaning to use the `InventorySettings.xml` file found in the default location to extend the functionality of the inventory component, copying it into the derived image) When this option is specified, `imgtrack` does not check for `InventorySettings.xml` or copy the file into the derived image. Unless the derived image already contains its own copy of the inventory settings file, this results in reduced capabilities for inventory gathering by `ndtrack.sh`. *Example:* `imgtrack example:latest --no-inventorysettings`

`--output-dir` *path* `-d` *path*	*Type:* Valid, existing directory on the host device (before triggering inventory collection, `imgtrack` verifies that the directory exists and that it can write to the directory, and exits with an error if verification fails) *Default:* Unset Copy the inventory to a directory on the host computer. When this option is given, a temporary directory is first created within the working directory used by `imgtrack`. The directory is mounted into the container at runtime, and the inventory is written into it. After the container terminates successfully, the inventory file is copied from the temporary directory into the final directory specified in this option. *Example:* `imgtrack example:latest -d /some/local/host/path`
`--pull`	*Type:* Boolean *Default:* Unset (false) Pull the identified image from the current registry of images to the local computer. For `imgtrack` to operate correctly, the image under investigation must be present in the local Docker image index on the local Linux device. This option causes `imgtrack` to run `docker pull` to transfer the image from the registry to the local image index before commencing any interactions with the image under investigation. To avoid setting authentication parameters on the command line, `imgtrack` does not support logging in to any registry. If the target registry requires authentication, an operator must either: • Use the docker `login` command prior to running `imgtrack` with the `--pull` option; or • Ensure that the target image is already present on the local machine prior to running `imgtrack`, and in this case the `--pull` option should be omitted. *Example:* `imgtrack example:latest --pull`
`--verbose` `-v`	*Type:* Boolean *Default:* Unset Enable more verbose logging from `imgtrack`. *Example:* `imgtrack example:latest -v`