



arbortext®

**Configuration Guide for  
Arbortext Publishing Engine**

7.1 M040

---

## **Copyright © 2019 PTC Inc. and/or Its Subsidiary Companies. All Rights Reserved.**

User and training guides and related documentation from PTC Inc. and its subsidiary companies (collectively "PTC") are subject to the copyright laws of the United States and other countries and are provided under a license agreement that restricts copying, disclosure, and use of such documentation. PTC hereby grants to the licensed software user the right to make copies in printed form of this documentation if provided on software media, but only for internal/personal use and in accordance with the license agreement under which the applicable software is licensed. Any copy made shall include the PTC copyright notice and any other proprietary notice provided by PTC. Training materials may not be copied without the express written consent of PTC. This documentation may not be disclosed, transferred, modified, or reduced to any form, including electronic media, or transmitted or made publicly available by any means without the prior written consent of PTC and no authorization is granted to make copies for such purposes. Information described herein is furnished for general information only, is subject to change without notice, and should not be construed as a warranty or commitment by PTC. PTC assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

The software described in this document is provided under written license agreement, contains valuable trade secrets and proprietary information, and is protected by the copyright laws of the United States and other countries. It may not be copied or distributed in any form or medium, disclosed to third parties, or used in any manner not provided for in the software licenses agreement except with written prior approval from PTC.

**UNAUTHORIZED USE OF SOFTWARE OR ITS DOCUMENTATION CAN RESULT IN CIVIL DAMAGES AND CRIMINAL PROSECUTION.**

PTC regards software piracy as the crime it is, and we view offenders accordingly. We do not tolerate the piracy of PTC software products, and we pursue (both civilly and criminally) those who do so using all legal means available, including public and private surveillance resources. As part of these efforts, PTC uses data monitoring and scouring technologies to obtain and transmit data on users of illegal copies of our software. This data collection is not performed on users of legally licensed software from PTC and its authorized distributors. If you are using an illegal copy of our software and do not consent to the collection and transmission of such data (including to the United States), cease using the illegal version, and contact PTC to obtain a legally licensed copy.

**Important Copyright, Trademark, Patent, and Licensing Information:** See the About Box, or copyright notice, of your PTC software.

### **UNITED STATES GOVERNMENT RIGHTS**

PTC software products and software documentation are "commercial items" as that term is defined at 48 C.F.R. 2.101. Pursuant to Federal Acquisition Regulation (FAR) 12.212 (a)-(b) (Computer Software) (MAY 2014) for civilian agencies or the Defense Federal Acquisition Regulation Supplement (DFARS) at 227.7202-1(a) (Policy) and 227.7202-3 (a) (Rights in commercial computer software or commercial computer software documentation) (FEB 2014) for the Department of Defense, PTC software products and software documentation are provided to the U.S. Government under the PTC commercial license agreement. Use, duplication or disclosure by the U.S. Government is subject solely to the terms and conditions set forth in the applicable PTC software license agreement.

PTC Inc., 140 Kendrick Street, Needham, MA 02494 USA

# Contents

About This Guide .....	7
Preparing to Configure Arbortext Publishing Engine .....	11
Arbortext Publishing Engine Tools and Resources .....	13
Components of Arbortext Publishing Engine .....	13
Initialization Process .....	18
Load Balancing and Clustering .....	20
Monitoring and Reporting Using a Web Browser .....	21
Configuring Arbortext Editor to Use Arbortext PE server .....	25
Publishing Configuration .....	26
Arbortext Publishing Engine Security Framework .....	27
Understanding Transactions on the Arbortext PE server .....	37
Transaction States .....	39
Using the Transaction Archive .....	40
Understanding Queuing on the Arbortext PE server .....	41
How Queuing Works .....	42
Configuring a Queue Manager .....	43
Configuring Queues .....	44
The Queued Transaction Scheduler .....	45
Queuing Query Parameters .....	46
Queuing for Arbortext Editor Clients .....	47
Monitoring Queues .....	49
Understanding Publishing Rules .....	51
Managing Publishing Rules .....	54
Deploying Publishing Rules .....	54
Windows Configuration .....	57
Configuring a User Account on Windows .....	58
Using the Arbortext Publishing Engine Configuration Program .....	60
Integrating Arbortext Publishing Engine with Apache Tomcat .....	66
Setting Configuration Parameters .....	69
The e3config.xml Configuration File .....	71
Arbortext Publishing Engine Global Parameters .....	71
Specifying a Request Handler .....	83
Specifying Request Selectors .....	89
Specifying Test Sets .....	91
Specifying Caches .....	92
Specifying Queues .....	93
Configuring Sub-process Pools .....	100

---

Specifying the AllowedFunctions List.....	107
Specifying Initializers .....	109
Requesting Administrative Information.....	111
Requesting a Status Report.....	112
Requesting a License Report.....	114
Requesting a Version Report .....	115
Monitoring the Transaction Archive .....	115
Requesting the Queue Reports.....	118
Requesting a Java Properties Report .....	121
Requesting Web Services Definitions.....	121
Requesting a Publishing Configuration Report .....	121
Usage Report .....	123
Requesting a Zip Archive for Troubleshooting .....	123
Requesting a Rescan of Publishing Configuration .....	123
Reloading Scripts .....	123
Running the Samples.....	124
Troubleshooting Arbortext Publishing Engine Operations .....	125
Using Arbortext Publishing Engine Interactive for Testing.....	126
Troubleshooting Publishing .....	126
Using the Arbortext Publishing Engine Test Utility.....	127
Troubleshooting Errors.....	127
Getting Trace Information.....	128
Publishing Issues.....	129
Reporting Problems to PTC Technical Support.....	130
Repository Connection Sample Script.....	131
Connecting to a Repository Adapter .....	132
Index.....	135

# Document Revision History

## 7.1 M040

<b>Section</b>	<b>Update</b>
Various	Added requirement to configure Apache Tomcat in line with current security best practices.





## About This Guide

This guide covers the configuration information you need to set up Arbortext Publishing Engine for your site. It contains information on the initial set up of configuration parameters. Then you can use it to tweak your site implementation and to aid in troubleshooting.

This *Configuration Guide for Arbortext Publishing Engine* manual provides information on configuring and managing the Arbortext PE Request Manager and its Arbortext PE sub-processes. It includes descriptions of archiving and queuing transactions, as well as monitoring and reporting capabilities for the Arbortext PE Request Manager and its Arbortext PE sub-processes. It also covers using Arbortext Publishing Engine as a publishing server for Arbortext Editor clients.

You can configure how to track transactions between clients and the Arbortext PE server. You can set logging and trace parameters to collect information when troubleshooting. You can save interim files produced during publishing, collect all document type files being used during transactions, and archive transactions.

Be sure to review *Arbortext Publishing Engine Release Notes* and *Installation Guide for Arbortext Publishing Engine* before you proceed with configuring Arbortext Publishing Engine. Also, *Arbortext Editor*, *Arbortext Publishing Engine*, *Arbortext Styler*, and *Arbortext Architect Release Notes* contains release information that may pertain to your use of Arbortext PE sub-processes, document types, stylesheets, and custom applications. Pay particular attention to publishing, API and other information that's not related to the user interface. Read it to get a complete view of release information before you complete your Arbortext Publishing Engine installation or upgrade.

Refer to [Components of Arbortext Publishing Engine on page 13](#) for a description of components specific to Arbortext Publishing Engine.

---

## Prerequisite Knowledge

This documentation is intended for the Arbortext PE server administrator. This document assumes advanced skill using servlet containers, web servers, and HTTP protocols. To install, set up, and configure Arbortext Publishing Engine, you should have substantial experience as a web server administrator. To understand the information in this manual, you should be familiar with the implementation at your site and with standard system administration tasks.

In a typical implementation, a client program or web browser sends an HTTP or SOAP request to a web server. The web server interprets the URL and passes the request to the servlet container. The servlet container knows how to call Arbortext PE Request Manager from its own configuration file, and it constructs and passes a request object and a response object to the Arbortext PE Request Manager. From the request object, the Arbortext PE Request Manager determines the client who sent the request, what work to perform, and what data to return in the response object to the servlet container. In turn, the servlet container returns the response to the web server, which then returns it to the client making the request.

A list of terms you should review:

- Request object — This is a Java object that contains all the information from a client HTTP or SOAP request. A servlet is configured to handle it by evaluating a portion of the URL for routing.
- Response object — This is a Java object that's created in conjunction with the Request object and constructed to handle the returned results. A custom application uses its methods to specify the information to be returned as the HTTP or SOAP response.
- Servlet — A web component, managed by a servlet container, that generates dynamic content. Servlets interact with web clients using a request and response model implemented by the servlet container. This request and response model is based on the behavior of the Hypertext Transfer Protocol (HTTP).
- Servlet container — A servlet container deploys the web application into its runtime environment. It loads and manages a servlet throughout its lifecycle using the servlet context. When the servlet container initializes the servlet, it creates the servlet context object which contains information about the servlet's runtime environment.
- Servlet context — A Java object that manages the runtime state of a servlet for the servlet container. It defines the servlet's view of the web application it's running in.
- SOAP — Formerly, an acronym for Simple Object Access Protocol, but now it's simply known as SOAP. SOAP is a cross-platform communication protocol based on XML that controls an exchange commonly consisting of a



---

request and a response. The data is exchanged in a SOAP message that usually uses HTTP as the transport mechanism. The server processes the SOAP message through a web service that the client calls when making a request.

For information about using SOAP with Arbortext Publishing Engine, see the *Programmer's Guide to Arbortext Publishing Engine*.

---

 **Caution**

There are potential security issues with SOAP and the Axis2 libraries. PTC recommends that you use HTTPS if you use SOAP. For more information, see *Arbortext Publishing Engine Security Framework* in the *Configuration Guide for Arbortext Publishing Engine*.

---

- **SOAP message** — The SOAP message is an XML document that follows a specific format. The client sends a request in a SOAP message to the web service, and the message contains a SOAP envelope. The SOAP envelope can consist of Header, Body (required), and Fault elements. The SOAP envelope supplies a requesting call to a server-side application through the web service. The SOAP envelope also includes instructions for returning a response.
- **Web application** — A collection of servlets, JavaServer Pages, HTML documents, images, archives, or other data.
- **WSDL** — An acronym for Web Services Description Language, which is a server-side document that describes the criteria by which a client application can call an application using SOAP. WSDL defines a SOAP web service by exposing its structure in XML format.



# 1

## Preparing to Configure Arbortext Publishing Engine

Arbortext Publishing Engine Tools and Resources .....	13
Components of Arbortext Publishing Engine .....	13
Initialization Process .....	18
Load Balancing and Clustering .....	20
Monitoring and Reporting Using a Web Browser .....	21

Before you begin setting up the Arbortext PE server, be sure you read the following:

- Review *Installation Guide for Arbortext Publishing Engine* to be sure Arbortext Publishing Engine was correctly installed, including integration with a servlet container or web server application.
- [Configuring Arbortext Editor to Use Arbortext PE server on page 25](#), if you will be configuring Arbortext Editor clients.
- [Understanding Transactions on the Arbortext PE server on page 37](#), if you will be managing transactions, especially using the transaction archive capability.
- [Understanding Queuing on the Arbortext PE server on page 41](#), if you will be implementing and managing queuing.
- [Windows Configuration on page 57](#)
- [Setting Configuration Parameters on page 69](#), especially [The e3config.xml Configuration File on page 71](#).

Before you proceed with Arbortext Publishing Engine configuration, you should be aware of the following:

- 
- The configuration files have predefined objects with either assigned or implicit default parameter values that enable much of Arbortext Publishing Engine functionality as installed. You will not need to change most of these values.
  - If someone at your site creates custom applications that implement any of the Arbortext Publishing Engine Java interfaces that will run on the server side of Arbortext Publishing Engine, you may need to create new object definitions, attributes, and parameters in the configuration files to allow them to run. The interfaces are described in *Programmer's Guide to Arbortext Publishing Engine*.

You'll need to work closely with the programmer implementing these applications for your site as the parameter list specified in the configuration files must meet the structure and requirements of the interface implemented in the custom application.

It's best to set up your server in a non-production area first until you have the configuration set up properly.

---

## Arbortext Publishing Engine Tools and Resources

The following describes specific Arbortext Publishing Engine tools and resources for configuring, developing, and testing your production environment.

### Tools and Resources

Component	Description
Arbortext Publishing Engine index page	HTML page that provides monitoring and reporting operations for Arbortext Publishing Engine and links for running test applications
Arbortext Publishing Engine status report	HTML page that displays a status report for Arbortext Publishing Engine and Arbortext PE sub-process activities
Arbortext Publishing Engine transaction archive	Archival tool that can be configured to keep transaction requests and responses, log messages, and archive the files used to process them.
Arbortext Publishing Engine Configuration program	Utility that controls some setup activities for Arbortext Publishing Engine
Arbortext Diagnostics program	Tool for tracing and logging Arbortext PE sub-process information, including the user account name for Arbortext Publishing Engine.
Arbortext Publishing Engine Test Utility	Tool for testing Java, JavaScript, VBScript, and ACL custom scripts
Arbortext Publishing Engine Interactive	User interface for the Arbortext PE sub-process used primarily for Arbortext Publishing Engine configuration, application development, and testing

## Components of Arbortext Publishing Engine

The following list describes the components of Arbortext Publishing Engine. Unless otherwise noted, Java programmers can implement their customized versions of these components.

- 
- **Arbortext PE Request Manager** — The Arbortext Publishing Engine Java servlet that handles all HTTP and SOAP requests and responses for document conversion and manipulation. It implements the standard `init`, `service`, and `destroy` methods of a servlet.

Incoming requests are passed to active Arbortext Publishing Engine Request Handlers according to configured criteria. When a response is returned to the servlet, the Arbortext PE Request Manager conveys the response to the servlet container, which returns the response to the client making the request.

- **Arbortext Publishing Engine Request Context** — A Java object that implements the **E3RequestContext** interface. The Arbortext Publishing Engine Request Context provides services and information about resources available to the context and writes to log files.

The Arbortext Publishing Engine Request Context object is created by the Arbortext PE Request Manager `init` method. It provides information to each Arbortext Publishing Engine Request Handler, Arbortext Publishing Engine Request Selector, Arbortext PE sub-process pool, Arbortext Publishing Engine Cache Manager, Arbortext Publishing Engine Queue Manager, and Arbortext Publishing Engine Initializer. The Arbortext Publishing Engine Request Context provides information about the Arbortext Publishing Engine environment such as parameter values, and services such as the location of an Arbortext PE sub-process service pool. It can also write to a servlet log and permits custom code to modify the global state of the Arbortext Publishing Engine environment.

- **Arbortext Publishing Engine Request Handler** — A Java object that implements the **E3RequestHandler** interface. The Arbortext PE Request Manager passes an incoming request to each defined Arbortext Publishing Engine Request Handler to determine which one is configured to service it.

A custom Arbortext Publishing Engine Request Handler must implement the `init`, `destroy`, and `service` methods. The `init` method is called during Arbortext PE Request Manager initialization, and it will receive the Arbortext Publishing Engine Request Context object and the parameters from the Arbortext Publishing Engine configuration file (`e3config.xml`). The `service` method is called to process a client request, creating a request object and a response object. The Arbortext PE Request Manager goes to each Arbortext Publishing Engine Request Handler, in the order they're defined, looking for a response that signifies whether it can process the query contained in the request. The Arbortext PE Request Manager calls the Arbortext Publishing Engine Request Handler's `destroy` method during termination.

- **Arbortext Publishing Engine Request Function** — A Java object that implements the **E3RequestFunction** interface. The Arbortext Publishing Engine Request Handler is configured to map query parameters from the

---

request to Arbortext Publishing Engine Request Functions. Each built-in Arbortext Publishing Engine function is mapped to a corresponding built-in Arbortext Publishing Engine Request Function. For example, the `f=status` function is mapped to the **com.arbortext.e3.FunctionStatus** interface.

The Arbortext Publishing Engine Request Handler distributed with Arbortext Publishing Engine can be customized to add custom function names and mappings without writing a new Arbortext Publishing Engine Request Handler.

- Arbortext Publishing Engine Request Selector — A Java object that implements the **E3RequestSelector** interface. Each Arbortext Publishing Engine Request Selector object applies test criteria to a request to help determine routing. If the request matches a predefined condition, the request is accepted by its Arbortext PE sub-process service pool for processing.

A Request Selector has an `init` and a `test` method. The `init` method is called during Arbortext PE Request Manager initialization; and it receives the information in the Arbortext Publishing Engine Request Context object and the parameters from the Arbortext Publishing Engine configuration file (`e3config.xml`). The `test` method is called when the Arbortext PE Request Manager needs to evaluate a request against a set of tests associated with an Arbortext PE sub-process pool. For example, a request selector could test a request for an HTTP header `content-type: text/xml` and return `TRUE` or `FALSE`.

Arbortext Publishing Engine Request Selectors can be logically combined using `AND` or `OR` in the Arbortext Publishing Engine Request Evaluator, otherwise known as the `TestSet` defined for each Arbortext PE sub-process pool.

- Arbortext Publishing Engine Request Evaluator — A Java object that implements the **E3RequestEvaluator** interface. The Arbortext Publishing Engine Request Evaluator can be defined in the `TestSet` section for each `SubprocessPool`, `QueueManager`, `Queue`, and `Notifier` defined in the Arbortext Publishing Engine `e3config.xml` file.

This wrapper-type `TestSet` object can combine a set of defined Arbortext Publishing Engine Request Selector tests to create test criteria. Two or more tests can be logically combined using `AND` and `OR` logic to form more complex criteria. For example, a `TestSet` might have a test to detect a particular `content-type` header and another test to detect a particular query parameter that also has a specific value. If an HTTP request meets this criteria, the Arbortext Publishing Engine Request Evaluator returns a boolean value of `TRUE` and the request would be accepted.

- 
- Arbortext Publishing Engine Initializer — A Java object that implements the **E3Initializer** interface.

This object's `init` method is called after all other objects are initialized. An initializer can execute startup tasks, such as retrieving and caching publishing configuration reports or transforming and caching stylesheets.

- Arbortext Publishing Engine Request Object — A Java object that implements the **E3ApplicationRequest** interface and contains all information being submitted in an HTTP or SOAP request to Arbortext Publishing Engine.
- Arbortext Publishing Engine Response Object — A Java object that implements the **E3ApplicationResponse** interface and contains the response to the HTTP or SOAP client making a request.
- Arbortext PE sub-process — An individual Arbortext Publishing Engine process, similar to Arbortext Editor, that performs document formatting, conversion, or other document manipulation for the Arbortext PE Request Manager as requested by an HTTP or SOAP client.
- Arbortext PE sub-process Pool — A Java object that implements the **com.arbortext.E3SubprocessPool** interface and controls a group of Arbortext PE sub-processes that are identical and interchangeable. The Arbortext PE Request Manager evaluates an HTTP or SOAP request against each Arbortext PE sub-process pool in the order they're defined in the Arbortext Publishing Engine `e3config.xml` configuration file.

Each Arbortext PE sub-process pool is uniquely identified and has its own parameters, such as minimum and maximum number of processes, time-out values, and so on. If you have more than one, each Arbortext PE sub-process pool must have an Arbortext Publishing Engine Request Evaluator `TestSet` defined in `e3config.xml` to specify the criteria for assigning HTTP requests to it.

One Arbortext PE sub-process pool must always be configured as the default pool; the default pool allocates an Arbortext PE sub-process to service any HTTP request if no other pool is configured or available. Because a request is assigned to the first Arbortext PE sub-process pool with matching criteria, the order in which pools are defined is important. Because the default pool is designed to service any HTTP request, put the default pool last to be sure that all other enabled pools are evaluated.

The Arbortext PE sub-process Pool can't be customized.

- Arbortext PE sub-process Context — A Java object that implements the **E3SubprocessContext** interface.

An Arbortext PE sub-process Context object contains any parameters specified in the `SubprocessContext` element for a `SubprocessPool`, along with the Arbortext PE sub-process pool name and the Arbortext PE sub-



---

process number (taken from the range of minimum to maximum number of Arbortext PE sub-processes).

Arbortext PE sub-process Context serves two purposes:

- An Arbortext Publishing Engine Request Handler can retrieve the information provided by the Arbortext PE sub-process Context.
- The Arbortext PE sub-process Context object is passed to the individual Arbortext PE sub-process taking the request, making it available to Arbortext Publishing Engine custom applications (ACL, Java, JavaScript, and VBScript).
- Arbortext Publishing Engine Queue Manager — Java object that implements the **com.arbortext.e3.QueueManager** interface.

An Arbortext Publishing Engine Queue Manager determines whether an HTTP request should be queued. The Arbortext PE Request Manager offers the request to each queue manager in the order in which they are defined in the Arbortext Publishing Engine configuration file. When a queue manager accepts the request, Arbortext PE Request Manager stops and takes the HTTP response from the Queue Manager and passes it to the client.

- Arbortext Publishing Engine Queue — Java object that implements the **com.arbortext.e3.ArbortextQueue** interface.

The Arbortext Publishing Engine Queue is a container of ordered transactions that are awaiting execution. Transactions are placed on a queue by an Arbortext Publishing Engine Queue Manager. The Queue Manager offers the request to each queue in the order in which they are defined in the Arbortext Publishing Engine configuration file. When a queue accepts the request, it holds it until the Queued Transaction Scheduler selects it for processing. After completion, a queued transaction is deleted from its queue.

- Arbortext Publishing Engine Notifier — Java object that implements the **com.arbortext.e3.E3Notifier** interface.

A notifier is called by the Arbortext PE Request Manager each time a transaction changes from one state to another. A notifier can be implemented and configured to respond to a transaction state change by performing an action, such as send an email.

- Arbortext Publishing Engine Transaction — A transaction is a request and response pair processed by the Arbortext Publishing Engine. The transaction includes the request transmitted to Arbortext PE Request Manager by a client, the response returned, and any intermediate files or logs generated when producing the response. The transaction is created upon receiving the request, and associated files are placed in the active transaction directory to wait for processing. There are two types of transactions:

- 
- an immediate transaction

Arbortext Publishing Engine transmits the response to the waiting client immediately after processing is complete. The immediate transaction will then be deleted from the active transaction directory. You can configure an immediate transaction to be archived in the Transaction Archive directory afterward.

- a queued transaction

Arbortext Publishing Engine places a transaction on a queue (provided the request meets queuing criteria) until it can be processed. After the transaction is processed, the client can retrieve the resulting output from the active transaction directory using the transaction ID. The transaction will be deleted from the active transaction directory according to the configuration settings for duration and retrieval by the client. You can configure a queued transaction to be archived in the Transaction Archive directory afterward.

## Initialization Process

During initialization, Arbortext Publishing Engine starts its minimum number of Arbortext PE sub-processes according to a configuration specification.

Periodically, Arbortext Publishing Engine checks for idle Arbortext PE sub-processes and tracks the lifetime duration of each Arbortext PE sub-process.

When an HTTP or SOAP request is passed to Arbortext PE Request Manager, Arbortext Publishing Engine assigns it to the idle Arbortext PE sub-process with the least elapsed time since last use. If there is no idle Arbortext PE sub-process, Arbortext Publishing Engine will start a new Arbortext PE sub-process to handle the request if the maximum number of Arbortext PE sub-processes allowed is larger than the minimum number. of Arbortext PE sub-processes.

Finding the right balance of configuration settings depends on your hardware, network traffic, system capabilities, volume and size of requests, and other factors that are particular to your site. In most cases, you will achieve the most efficient throughput by setting the maximum number of Arbortext PE sub-processes to the number of processors on the Arbortext Publishing Engine server system. This guideline ensures all CPUs will be used and avoids unnecessary context switching, which can hinder throughput.

## Arbortext PE Request Manager Startup

Like every Java Servlet, the Arbortext PE Request Manager provides three methods that are called by the Java servlet container in which it runs: `init`, `service`, and `destroy`.

---

The Arbortext PE Request Manager's `init` method is called by the Java servlet container, either when the container itself starts or when the container receives the first request for Arbortext Publishing Engine. During initialization, the Arbortext PE Request Manager will perform the following functions:

1. Locates and reads the `e3config.xml` configuration file and constructs an `E3ConfigFile` object containing defined elements, their attributes, and their parameters.
2. Creates the Arbortext Publishing Engine Request Context that provides initialization information to all the Arbortext Publishing Engine Java objects specified in the `e3config.xml` file.
3. Starts and initializes each Java object, using its parameters described in `e3config.xml`, such as request selector tests, cache managers, queue managers, request handlers, subprocess pools, and initializers. Generally speaking:
  - Each object described in `e3config.xml` must implement a supported Arbortext Publishing Engine Java interface with an `init` method.
  - The Arbortext Publishing Engine Request Context object is passed to each `init` method.
  - Objects are initialized in the order in which they appear in `e3config.xml`.
  - As each defined object is initialized, the Arbortext PE Request Manager adds knowledge about that object to its data structure, as acquired through the Arbortext Publishing Engine Request Context object.

After initialization, HTTP requests can be processed. The servlet container calls the Arbortext PE Request Manager's `service` method each time an HTTP request is received. The `service` method performs the following operations:

1. Passes the request to each Arbortext Publishing Engine Cache Manager, in the order they're defined in `e3config.xml`. If a cache manager has a cached response that can satisfy the request, the response is returned to the client and the `service` function returns.
2. Passes the request to each Arbortext Publishing Engine Queue Manager, in the order they're defined in `e3config.xml`. If a Queue Manager is willing to accept the request for processing later, the Queue Manager sets the response to be returned to the client and stores the request. Then the `service` function returns.
3. Passes the request to each Arbortext Publishing Engine Request Handler, in the order they're defined in `e3config.xml`. The Request Handler

---

determines if it can process the request or if the request needs an Arbortext PE sub-process.

4. If the request requires an Arbortext PE sub-process, the Request Handler will pass it to each Arbortext PE sub-process pool, in the order they're defined in `e3config.xml` until one can accept it for processing.

## Load Balancing and Clustering

Load balancing and clustering can improve scalability, availability, fail-over capabilities and performance.

- Load balancing distributes the work load to multiple identical servers based a predefined load-balancing policy. It can be implemented using hardware, software, or a combination of both. Two popular methods use DNS round robin distribution or server hardware to direct network traffic using some kind of network device.

Choosing a load balancing approach depends on existing infrastructure, cost, and demand, among other considerations. You will need to research specific hardware and software solutions for implementing load balancing.

- Clustering is implemented as software installed on a group of servers organized into clusters. It shares the work load according to the software protocol.

Clustering can support features not available with server load balancing, such as persistent session data across multiple server nodes. Clustering can be more difficult to implement.

Arbortext Publishing Engine can be deployed using either a server load balancing approach or clustering, except for those that use web farms. Arbortext Publishing Engine should not be clustered in a group of virtual servers running on a single physical server.

It's easier to implement load balancing for Arbortext Publishing Engine as it does not specifically need the session data used in clustering or sticky routing.

Whenever Arbortext Publishing Engine is deployed on multiple physical servers, regardless of the method you choose, all servers running Arbortext Publishing Engine must have completely identical installation trees with all customizations in place.

In Apache Tomcat, load balancing can be implemented by directing requests to the provided servlet Balancer. The Balancer servlet requires some Java code rules to implement a useful load balancing scheme. Refer to the Balancer servlet documentation for more information. In Tomcat, clustering requires programming

---

Java code, so its implementation is more difficult. Choose clustering only if you need the additional benefits for other reasons, as they are not used by Arbortext Publishing Engine.

## Modifications Made by Requests

When using load balancing or clustering, it is important that requests do not rely on modifications made by previous requests. For example, a request could run a script that modifies a `set` command option. The `set` command option will be modified only on the Arbortext PE sub-process running on the Arbortext PE server handling the request. A request to Arbortext Publishing Engine can't rely on any action performed by a previous request. Requests are fulfilled by multiple Arbortext PE sub-processes that do not maintain a session state or record of a previous request. Subsequent requests could go to a different Arbortext PE sub-process or a different Arbortext PE server. This means that Arbortext PE Request Manager has no information to determine how to route subsequent requests based on previous ones.

## Using the Administration Tools

Arbortext Publishing Engine is deployed by installing it on each server in a group. Usually, a central server handles the distribution load, and it is the only server identified to clients for their use. The identity of other servers in the group may only be known to administrators. For an administrator to access the functionality available for Arbortext Publishing Engine, a request must be made to the index page on each server where Arbortext Publishing Engine is installed. Accessing the index page on the central server distributes the request to an unidentified, subordinate server. This means that you need to access each server individually to obtain Arbortext Publishing Engine reports, monitor activity, and review or collect archived transaction files for troubleshooting. Refer to [Monitoring and Reporting Using a Web Browser on page 21](#) for information about the features available from the Arbortext Publishing Engine index page.

## Monitoring and Reporting Using a Web Browser

Arbortext Publishing Engine has an index page with links that return information and perform a variety of administrative actions and sample document conversions. After you've successfully installed and configured Arbortext Publishing Engine, this index page is available from a web browser. Use a URL that follows the example:

```
http://servername:port/e3/
```

In the URL, *servername* is the name of the Arbortext PE server machine, and *port* is the port number the servlet container or web server uses to monitor HTTP requests on its behalf.

### View PTC Arbortext Publishing Engine Information

<a href="#">Status, License, Version</a>	Display current status, version, or license information.
<a href="#">Transaction Archive</a>	Display information about completed transactions.
<a href="#">Queue List</a>	Display information about queues and queued transactions.
<a href="#">Java Properties</a>	Display the properties of the JVM in which PE is running.
<a href="#">Web Service Definitions</a>	View the WSDL file for PE web services.
<a href="#">Short, Detailed, Log</a>	View information about the publishing configuration.
<a href="#">Usage Report</a>	Display a report summarizing how client machines are using PE.
<a href="#">Application Save</a>	Obtain a zip archive describing PE Subprocess configuration.
<a href="#">All Available Information</a>	Obtain a zip archive containing all of the above information.

### Administer PTC Arbortext Publishing Engine

<a href="#">Rescan Publishing Configuration</a>	Rebuild the publishing configuration document used by PTC Arbortext Editor clients.
<a href="#">Reload Subprocesses</a>	Reload cached stylesheets and script-based applications used by PE subprocesses. Java applications will not be reloaded.

### Test PTC Arbortext Publishing Engine

<a href="#">EPUB, HTML, HTML Help, PDF, PostScript, RTE, SGML, XML</a>	Convert an XML document (e3demo.xml) to the specified output format. (Graphic links may not work correctly.)
<a href="#">DMP Input, Image, Web, or Help; Web Application</a>	Convert an XML document (e3demo.xml) to a zip archive containing the specified format.
<a href="#">Word, RTE, HTML, PDF, Text, DocX, WordML</a>	Convert a sample file in the selected format to XML.
<a href="#">ACL, Java, JavaScript, VBScript</a>	Run a test PE application.

## View Arbortext Publishing Engine Information

- The `Status` link returns a status report. It includes:
  - basic installation, system, COM, allowed functions and global parameters information.
  - Arbortext PE sub-process pool status, including whether it's enabled or disabled, its associated configuration settings, the process IDs and allocation status of each Arbortext PE sub-process.
  - all configuration settings for caches, queues, request handlers, and request selectors.
  - information on the Queue List, if queues are configured. Information about individual queues is available from the Queue List page.
  - system **Environment Variables**. If `PTC_D_LICENSE_FILE` is set, it will be included in the **Environment Variables** report.
- The `License` link retrieves basic information about the installation, as well as the license source, the user under which Arbortext Publishing Engine is

---

running, and the number of processor cores and packages on the Arbortext PE server. It also lists the optional software components installed and whether they are licensed.

A license error report will be returned with the information that *PTC\_D\_LICENSE\_FILE* is either missing or set to an incorrect value (and the incorrect value will be reported).

- The `Version` link retrieves version information about Arbortext Publishing Engine and its Arbortext PE sub-processes.
- The `Transaction Archive` link retrieves information on the transaction archive, if one has been implemented. More information about individual archived transactions is available from the Transaction Archive page.
- You can retrieve a report on the `Queue List`, if queues are configured. Information about individual queues is available from the Queue List page.
- The `Java Properties` link returns all information about the JVM.
- The `Web Service Definitions` link returns the Arbortext Publishing Engine WSDL definitions.
- You can retrieve any of three variations of the Publishing Configuration report.
- You can retrieve a `Usage Report` with a summary of clients and transactions, and usage by client.
- You can run an `Application Save` to retrieve configuration about Arbortext PE sub-processes.
- `All Available Information` returns a zip archive containing all the information available about Arbortext Publishing Engine listed in this section, as well as the output from the sample PE applications in the testing section and the `Application Save` zip archive.

## Administer Arbortext Publishing Engine

You can rescan the publishing configuration information and its cache for use by Arbortext Editor clients.

You can reload scripts and update cached stylesheets on Arbortext PE server for use by all clients.

## Test Arbortext Publishing Engine

You can convert a sample document to any of the supported output formats listed.

---

You can run Arbortext Publishing Engine sample test applications which return basic information about server configuration. The source code for these sample applications is available from your installation in `PE_HOME\e3\samples`, and they're described in the *Programmer's Guide to Arbortext Publishing Engine*.

None of the actions available from the Arbortext Publishing Engine index page are queued.



# 2

## Configuring Arbortext Editor to Use Arbortext PE server

Publishing Configuration .....	26
Arbortext Publishing Engine Security Framework.....	27

Arbortext Editor users can be set up to use Arbortext Publishing Engine to publish their documents. For information on the specific versions of Arbortext Editor clients supported by Arbortext Publishing Engine, consult the system and software requirements section of the *Installation Guide for Arbortext Publishing Engine*.

Arbortext Editor users need to ask the Arbortext Publishing Engine administrator for the URL that specifies the server, port number, and Arbortext Publishing Engine servlet specification as configured by the web application server or servlet container where Arbortext Publishing Engine is installed. In a standard installation, Arbortext Editor users have the option to enter the URL for the Arbortext PE server. These options are also available to Arbortext Editor users from the **Publishing Engine** panel of **Tools ▶ Preferences**.

If your site will be deploying a compact installation of Arbortext Editor, be sure to consult the Arbortext Editor Deployment Kit Guide. The compact deployment recommends adding `set peservices` and `set peserverurl` options to the Deploy Directory's `siteprefs.xml` file.

Arbortext Editor users can request a report containing Arbortext Publishing Engine publishing information, such as document types, stylesheets, framesets, and content pipelines available from Arbortext Publishing Engine when publishing documents. On the Arbortext Editor client, go to the **Help ▶ PE Configuration** command.

---

## Publishing Configuration

The configuration information returned in the Arbortext Publishing Engine Publishing Configuration report lists all the document types, document type configuration files, and stylesheets available from the Arbortext PE server. You can review this information by choosing the link for the format you want for **View information about the publishing configuration** from the Arbortext Publishing Engine index page. Click the **Short** link to retrieve the HTML version of the report. (Refer to [Monitoring and Reporting Using a Web Browser on page 21](#) for more information on using the index page.)

The PE Publishing Configuration report lists all available stylesheets, and it references a stylesheet by both its path and its name in separate entries. The report warns of any duplicate stylesheet names on the server. If a stylesheet name is not unique on the server, Arbortext Publishing Engine uses the first one it finds.

If Arbortext Publishing Engine is restarted, the Arbortext Editor client is not aware of any changes. To be certain Arbortext Editor has up-to-date publishing information, users must perform one of the following to cause Arbortext Editor to automatically obtain the latest Arbortext Publishing Engine publishing configuration information:

- Disable and then enable **Use Publishing Engine** in the **Publishing Engine** panel of **Tools ▶ Preferences**.
- Turn `peservices` off and then on again in **Tools ▶ Preferences ▶ Advanced**.
- Exit and restart Arbortext Editor.

Arbortext Editor users can take advantage of the Arbortext Publishing Engine queuing capabilities. For more information, refer to [Queuing for Arbortext Editor Clients on page 47](#).

Arbortext Editor can also compare its publishing configuration with Arbortext Publishing Engine publishing configuration using the **Tools ▶ Compare Config with PE** menu item. In the **Compare Publishing Configuration with PE** dialog box, Arbortext Editor users can generate the Publishing Configuration Comparison report, which notes the differences between the publishing environment on the client and on the server. This report is helpful in troubleshooting publishing processing, and you should include it with the data you submit when reporting a problem to Technical Support.

The Arbortext Editor user has a companion tool to use for troubleshooting. If there are problems with publishing, the user can run a **Tools ▶ Save Application** and choose from two **Save** options for gathering data about publishing:

- **Local Data**  
saves the application data associated with the local machine's publishing configuration, and it also enables **Compare Local Configuration with PE**. If it's checked, the application save will contain the comparison of the local

---

machine's publishing configuration with the Arbortext PE server configuration.

- **PE Data**

transmits the current document to the Arbortext PE server. The server opens the document and generates an application save on the server. It will include the data requested from the choices for **Copy document type** and **Copy all applications**.

---

 **Note**

If you are using a content pipeline on the Arbortext PE server for processing large documents and have memory consumption problems, you can improve content pipeline publishing processing by using the `doc_estimate_dfs` ACL function and `set bigjobthreshold` ACL command option. Refer to the *Arbortext Command Language Reference* for information.

---

## Arbortext Publishing Engine Security Framework

Arbortext Publishing Engine includes a security framework that allows every request sent to the Arbortext PE Request Manager to be classified as "disabled", "unrestricted", or "restricted" based on users and groups defined in Apache Tomcat.

- Disabled requests are not processed.
- Unrestricted requests are processed without authentication.
- Restricted requests are only processed if they are submitted by an authenticated user who is a member of a configured security role.
  - Authenticated requests submitted by other users are not processed.
  - If a client submits an unauthenticated request, the Arbortext PE Request Manager will reject the request in a way that instructs the client to prompt for an ID and password and resubmit the request. If the client authenticates successfully, and the user is a member of the required role, then the request will be processed.

If a request cannot be processed, the Arbortext PE Request Manager will return an error message in the HTTP response returned to the client. For every request received, the Arbortext PE Request Manager will write a line to an audit file explaining why the request was processed or not processed.

---

 **Note**

While enabling the Arbortext Publishing Engine security framework provides a layer of security against improper access to Arbortext Publishing Engine, it should be considered as only one component of your site's broader security plan.

---

The security framework is enabled, disabled, and configured using entries in `e3config.xml`. If the framework is disabled, none of the described request processing takes place and the Arbortext PE Request Manager will operate as it did in earlier versions of PE. When installing Arbortext Publishing Engine, the user is prompted to enable the framework. If the user chooses to enable the framework, `e3config.xml` will be updated accordingly. By default, the framework is disabled.

The security framework makes use of the user ID and role support provided by Apache Tomcat. Tomcat supports defining user IDs, securing each user ID by a password, and mapping each user ID into one or more roles. The Arbortext Publishing Engine security framework makes use of this support to determine whether a restricted request should be processed or rejected.

The following sections detail how to enable and configure the framework, and provide the requirements for configuring Apache Tomcat to work with the Arbortext Publishing Engine security framework.

---

 **Note**

You must also ensure that Tomcat is configured in line with current security best practices.

---

## Enabling the Security Framework

During installation of Arbortext Publishing Engine, you can choose to enable the Arbortext Publishing Engine security framework. However, Apache Tomcat must be properly configured before enabling the Arbortext Publishing Engine security framework. If the security framework is enabled without Tomcat being already properly configured, Arbortext Publishing Engine will not function properly.

Once Arbortext Publishing Engine is installed and the security framework and Tomcat are properly configured, enable the security framework by opening `e3config.xml` and setting the `com.arbortext.e3.enableSecurityFramework` parameter to a value of `true` as in the following example.

---

```

<!-- This parameter specifies whether the security framework
is enabled or disabled. For compatibility with versions
of PE earlier than 7.0 M030, set it to "false".
CAUTION: if this value is set to "false", PE will allow
any user to submit any request. No authentication will
be performed.
-->
<Parameter name="com.arbortext.e3.enableSecurityFramework"
value="true"/>

```

Restart Arbortext Publishing Engine for the change to take effect.

## Configuring Security Constraints

Security constraints are the mechanism used by the Arbortext PE Request Manager to classify a request as disabled, unrestricted, or restricted. Each security constraint consists of one or two parameters and a test set. When the Arbortext PE Request Manager receives a request, it searches the list of security constraints defined in `e3config.xml` until it finds a security constraint with a test set matching the request. The search is in the order security constraints are configured in `e3config.xml`. If no security constraint matches the request, the Arbortext PE Request Manager rejects the request with an error message.

If a security constraint's `access` parameter value is `restricted`, the Arbortext PE Request Manager checks to see if the request is authenticated. If not, the Arbortext PE Request Manager rejects the request using an HTTP result code that states authentication is required. Most web browsers will prompt for an ID and password and resubmit the request. If the password is legal for the ID, the resubmitted request will be considered properly authenticated. If the request is authenticated, it will have an associated user ID and the Arbortext PE Request Manager will check to see whether the user ID has the role specified by the security constraint's `role` parameter value. If the ID has the role specified, the request is allowed to proceed. If the role does not, the Arbortext PE Request Manager rejects the request with an error message.

Following is a sample security constraint configuration as delivered in `e3config.xml`. Note that the security constraint named "admin-requests" refers to the standalone test set "admin-tests" using the `ref` attribute, while security constraint named "unrestricted-requests" has an in-context test set.

```

<!-- Security Constraints that determine who can submit which
requests. Note that users and roles are defined in web.xml
and in Tomcat's tomcat-users.xml configuration files.
-->
<SecurityConstraints>
  <!-- The following requests must be submitted by an
authenticated user with the "pe-admin" role. -->
  <SecurityConstraint id="admin-requests">
    <Parameter name="access" value="restricted"/>
    <Parameter name="role" value="pe-admin"/>

```

```

<TestSet ref="admin-tests"/>
</SecurityConstraint>
<!-- The following requests may be submitted by any user.
No authentication is required. -->
<SecurityConstraint id="unrestricted-requests">
<Parameter name="access" value="unrestricted"/>
<TestSet>
<Or>
<Test name="test-request-unrestricted"/>
<Test name="test-editor-queuing"/>
</Or>
</TestSet>
</SecurityConstraint>
</SecurityConstraints>

```

Creating security constraints can become fairly complex, and it is possible to code an incomplete or erroneous security constraint in `e3config.xml`. For example, a constraint might not have an `access` parameter, a constraint with an `access` parameter value of `restricted` might not have a `role` parameter defined, or the test set might reference a non-existent standalone test set. The Arbortext PE Request Manager will check for such inconsistencies when it initializes and reads `e3config.xml`. For each error it finds, it will make an entry in its error log. Be aware of the following items:

- If a constraint's test set does not exist, or contains tests that are invalid, the constraint will match all requests.
- If the Arbortext PE Request Manager detects any error, the security constraint will behave as if it has an `access` parameter value of `disabled`. This will result in any request that the security constraint handles being rejected, with errors returned to the client and written to the servlet log.

Security constraints are described on the Arbortext Publishing Engine status report. The entry for any inconsistent constraints will state that the constraints are inconsistent.

## Configuring Users and Roles

### Configuring Roles on Arbortext Publishing Engine

At installation, `e3config.xml` references only one role: "pe-admin". You may want to define more roles. (For example, restricting `f=java` requests to users with the role "java-client" and `f=acl` requests to users with the role "acl-client".)

---

To define additional roles, perform these steps:

1. Open `e3config.xml` and create security constraints with `role` parameters specifying the new roles.
2. Open `PE_HOME\e3\WEB-INF\web.xml` and define each new role created in `e3config.xml`. The content of `web.xml` is defined in the Java Servlet Standard.

Following is an example of the role-related content of `web.xml` as installed:

```
<!-- These are the security roles referenced by e3config.xml.
  If you need additional roles in e3config.xml, you must add
  them here, too. Note that you must modify Tomcat's tomcat-
  users.xml file to define user IDs which are members of these
  roles.
-->
<security-role>
  <role-name>
    pe-admin
  </role-name>
</security-role>
<security-role>
  <role-name>
    pe-user
  </role-name>
</security-role>
```

(The role `pe-user` is not used in `e3config.xml`, but is included as an example.)

## Configuring Users and Roles in Apache Tomcat

You must configure Apache Tomcat with the user IDs and roles that Arbortext Publishing Engine will be checking for. Configure roles and users in the file `conf\tomcat-users.xml`, found in the top-level Tomcat installation directory. Apache Tomcat is installed with a `tomcat-users.xml` that contains only comments. No roles or users are defined by default.

Follow is an example of roles and users as they could be defined in `tomcat-users.xml`:

```
<tomcat-users>
<role rolename="pe-admin"/>
<role rolename="pe-user"/>
<user username="ptc" password="ptcpassword" roles="pe-admin"/>
<user username="user" password="userpassword" roles="pe-user"/>
</tomcat-users>
```

---

**⚠ Caution**

Do not use the configuration as shown here in your production system. Unauthorized people trying to access your Arbortext PE server may try these published values.

You must also ensure that Tomcat is configured in line with current security best practices.

---

This defines two roles and two user IDs, each of which is a member of one role. Configuring with this example can be useful for testing. If you use a web browser to access the Arbortext Publishing Engine Index Page and are asked for authentication, authenticate as user "ptc" and you should be allowed access. Authenticate instead as "user" and access should be disallowed.

---

**💡 Note**

Defining users and roles in `tomcat-users.xml` is supported by Apache Tomcat as installed. However, defining a site's users and roles in this manner may not be the best solution for many environments. For example, you may wish to have your users IDs and passwords be defined by your existing Tomcat-based corporation security infrastructure. Refer to the documentation supporting your existing system for information on its integration with other systems.

---

## Security Framework Logging

Each time the Arbortext PE Request Manager receives a request and evaluates it, it writes an entry to an audit log file describing the request itself, the decision reached (allow to proceed, reject), and the reason for the decision. Following is an example from an audit log file:

```
09:27:20 [http-apr-8080-exec-2] INFO GateLogEntry.auditLog -
  uri='/e3/servlet/e3' secured='false' host='127.0.0.1'
  addr='127.0.0.1' protocol='HTTP/1.1' scheme='http' method='GET'
  query='f=status' sc='admin-requests' authReq='true'
  alreadyAuth='false' triedAuth='true' rcAuth='false'
  remoteUser=null' allowed='false' why='du' status='401'
  reason='Response set up to request authentication.'
09:27:25 [http-apr-8080-exec-3] INFO GateLogEntry.auditLog -
  uri='/e3/servlet/e3' secured='false' host='127.0.0.1'
  addr='127.0.0.1' protocol='HTTP/1.1' scheme='http'
  method='GET' query='f=status' sc='admin-requests'
  authReq='true' alreadyAuth='false' triedAuth='true'
```



```

rcAuth='true' remoteUser=ati' allowed='true' why='arm'
status='200' reason='user 'ati' has role 'pe-admin''
10:53:47 [http-apr-8080-exec-6] INFO GateLogEntry.auditLog -
uri='/e3/jsp/queuelist.jsp' secured='false' host='127.0.0.1'
addr='127.0.0.1' protocol='HTTP/1.1' scheme='http' method='GET'
query='null' sc='admin-requests' authReq='true'
alreadyAuth='false' triedAuth='true' rcAuth='false'
remoteUser=null' allowed='false' why='du' status='401'
reason='Response set up to request authentication.'
10:53:47 [http-apr-8080-exec-7] INFO GateLogEntry.auditLog -
uri='/e3/jsp/queuelist.jsp' secured='false' host='127.0.0.1'
addr='127.0.0.1' protocol='HTTP/1.1' scheme='http' method='GET'
query='null' sc='admin-requests' authReq='true'
alreadyAuth='false' triedAuth='true' rcAuth='true' remoteUser=ati'
allowed='true' why='arm' status='200'
reason='user 'ati' has role 'pe-admin''
10:53:52 [http-apr-8080-exec-5] INFO GateLogEntry.auditLog -
uri='/e3/servlet/e3' secured='false' host='127.0.0.1'
addr='127.0.0.1' protocol='HTTP/1.1' scheme='http' method='GET'
query='f=app&file=$aptpath/e3/e3/e3demo.3f'
sc='unrestricted-requests' authReq='false' alreadyAuth='false'
triedAuth='false' rcAuth='false' remoteUser='null' allowed='true'
why='aru' status='200'
reason='No authentication required for this request.'
```

Each entry starts with the time of the request, the thread ID, the message level, and the issuing module. Each entry then reports the following items:

- uri — URI of the request as received by Arbortext Publishing Engine
- host — Name of the Arbortext PE server
- addr — IP address of the Arbortext PE server
- protocol — Request protocol
- scheme — Scheme of the URL (http or https)
- query — The request query string
- sc — ID of the security constraint that matched the request
- authReq — "true" if authentication was required. Otherwise, "false".
- alreadyAuth — "true" if the request was already authenticated
- triedAuth — "true" if the Arbortext PE Request Manager tried to authenticate the request
- rcAuth — "true" or "false" as returned from the authentication attempt
- remoteUser — User ID of an authenticated request
- allowed — "true" if the security constraint allowed the request to proceed. Otherwise, "false".
- why — Value to provide to PTC Technical Support when filing a case

- 
- `status` — HTTP result code returned if access was denied
  - `reason` — Description of why access was or was not allowed

Use the following approaches to ensure requests are being properly accepted and rejected.

- Examine the audit log file.
- Open `e3config.xml` and set the `debug` flag to “true”. Send requests to Arbortext Publishing Engine and examine the servlet log.

## Customizing the Security Framework

When customizing the Arbortext Publishing Engine security framework, be aware of the following items.

- By default the Arbortext Publishing Engine security framework divides all of the requests that Arbortext Publishing Engine processes into 2 classes: administrative and non-administrative. The former processes are restricted, the latter are unrestricted. The framework can be made more precise by adding additional security constraints that are more finely grained; matching individual `f=acl` requests, `f=java` requests, and so on, rather than matching all requests.
- Some requests that Arbortext Publishing Engine supports should not be made restricted, because they are submitted by client programs that cannot process requests for IDs and passwords. For Arbortext Publishing Engine composition (in which Arbortext Editor submits publishing requests to the Arbortext PE Request Manager, the following requests must remain unrestricted:
  - `f=java class=com.arbortext.e3ci.Application`
  - `f=java class=com.arbortext.e3c.Application`
  - `f=qt-cancel`
  - `f=qt-discard`
  - `f=qt-list`
  - `f=qt-retrieve`

The final four `f=qt-` functions are only required if Arbortext Editor is submitting `QUEUED` transactions. If your site doesn't use queuing, you can restrict these four functions and disable queuing by setting the parameter `com.arbortext.e3.queueCompositionOperations` to `never`.

- For composition requests from the WVS Arbortext Publishing Engine Worker, the following request must remain unrestricted:  
`f=java class=com.arbortext.ptc.windchill.Compose`

- 
- For composition requests from the Windchill Service Information Manager Worker (also called the SIS Worker), the following request must remain unrestricted:  
`f=acl function=main::composeSisPE`
  - Be aware that, in addition to the controlling the security framework, the `e3config.xml` file restricts the ACL, APP, Java, JavaScript, and VBScript PE applications that can run.
  - Besides configuring Arbortext Publishing Engine, several items can be configured in Apache Tomcat by modifying `web.xml` or Tomcat's `servlet.xml`.

In `servlet.xml`, you can configure the following items.

- Enable HTTPS with or without client-side certificate authentication
- Disable HTTP so that only HTTPS requests are accepted
- Control how session IDs are managed

For more information, refer to the documentation provided for Tomcat by the Apache Software Foundation.

In `web.xml`, you can configure the following items.

- Session timeouts
- The authentication method used
- How cookies are used
- Additional security roles

For more information, refer to the Java Servlet Standard.

---

 **Note**

You must also ensure that Tomcat is configured in line with current security best practices.

---



# 3

## Understanding Transactions on the Arbortext PE server

Transaction States .....	39
Using the Transaction Archive .....	40

A transaction is a request transmitted to Arbortext PE Request Manager by a client, the response returned after processing, and any intermediate files or logs generated in the process of producing the response. After receiving a request, Arbortext PE Request Manager assigns a unique transaction ID, provides an optional transaction name if supplied, and allocates an associated Transaction Directory for the request. Arbortext Publishing Engine stores the data associated with the request in that directory. As it processes the request and generates a response, Arbortext Publishing Engine writes additional control information, intermediate files, and the response itself to the transaction directory.

A transaction is considered complete when the process is finished and a result is generated. The result of a complete transaction can be successfully published output or instead, contain errors or warnings, as long as the transaction finished.

If the request is an immediate transaction, meaning it will be processed as soon as possible, Arbortext Publishing Engine transmits the response to the waiting client immediately after processing is complete. The immediate transaction will then be deleted from the active transaction directory. However, you can configure the transaction to be archived in the Transaction Archive directory afterward.

If the request is a queued transaction, meaning Arbortext Publishing Engine places the transaction on a queue, it will wait in a queue until it can be processed according to the queuing configuration criteria. The associated files are placed in the active transaction directory to be available when the transaction is processed. After the transaction is processed, the client can retrieve the resulting output from the active transaction directory using the transaction ID. The transaction will be

---

deleted from the active transaction directory according to the configuration settings for duration and retrieval by the client. Arbortext Editor users retrieve the output using **Tools ► Queued Transactions**.

When a transaction is completed, expires, or is deleted from the active transaction directory, the Arbortext PE Request Manager determines if the transaction meets the criteria for archiving in a Transaction Archive directory afterward. A public Transaction Archive can be visible to users from the Arbortext PE server index page. If a particular transaction is considered secure, an alternate transaction archive can be configured to store it, where it will be unavailable through the index page. You can configure transaction archive parameters in `e3config.xml`.

Arbortext PE Request Manager tracks the transaction ID so that it's not reused, but it has no knowledge about whether the transaction has been archived. The parameters that control transaction handling and storage are described in [The Global Active Transaction Parameters on page 71](#) and the [The Global Transaction Archive Parameters on page 74](#). To learn about queuing, refer to [Understanding Queuing on the Arbortext PE server on page 41](#). Transactions can also be named, supplied in the submitted query from an application or Arbortext Editor, or configured using a global parameter. Refer to [Queuing Query Parameters on page 46](#) and [The Global Transaction Name Parameter on page 73](#) for more information.

You can track queued transactions that are waiting to be processed or are completed. See [Monitoring Queues on page 49](#) for more information.

---

## Transaction States

A transaction can exist in one of a number of distinct states during its lifecycle.

- `initializing`  
The Arbortext PE server is receiving the request.
- `waiting`  
The transaction is waiting to be allocated to an Arbortext PE sub-process (for immediate requests).
- `queued`  
The transaction was placed in a queue and is waiting to be executed by the Queued Transaction Scheduler.
- `processing`  
The transaction is being executed by an Arbortext PE sub-process.
- `+`  
`complete`  
The transaction is finished. A completed transaction can be completed successfully or completed with errors. Note that if the results is an error report rather than the expected document, the transaction is still considered complete.  
A transaction can also expire while waiting for an Arbortext PE sub-process allocation (if it's an immediate transaction).
- `cancelled`  
The transaction has been cancelled.

The transaction lifecycles are as follows:

- An immediate request transaction follows the lifecycle:  
Initializing ⇒ Waiting ⇒ Processing ⇒ Complete
- A queued request transaction follows the lifecycle:  
Initializing ⇒ Queued ⇒ Processing ⇒ Complete or Cancelled

A completed transaction may be available in the Transaction Archive if it meets the archiving criteria. Refer to [Using the Transaction Archive on page 40](#) for more information.

## Transaction Notifiers

When a queued transaction changes from one state to another, it can inform a configured notifier object of the change. A notifier object can be called by the Arbortext PE Request Manager each time a queued transaction changes from one

---

state to another. Notifiers are configured in `e3config.xml`. A notifier accepts parameters that control its operation, as well as an optional `TestSet` to respond only to certain types of transaction states. A notifier can be configured to take an action based on a change in queued transaction state. These states can be specified in the notifier parameter `com.arbortext.e3.target-states`, described in [Configuring a Notifier on page 97](#).

## Using the Transaction Archive

The Transaction Archive can store all files used in processing a transaction after it's finished. The parameters that control the behavior for archiving transactions are described in [The Global Transaction Archive Parameters on page 74](#).

The Transaction Archive lists the transactions that have been processed and subsequently archived on the Arbortext PE server. This report is available from the **Transaction Archive** link on the Arbortext Publishing Engine index page. The report displays the transactions from newest to oldest. Refer to [Monitoring the Transaction Archive on page 115](#) for information.



# 4

## Understanding Queuing on the Arbortext PE server

How Queuing Works.....	42
Configuring a Queue Manager.....	43
Configuring Queues.....	44
The Queued Transaction Scheduler.....	45
Queuing Query Parameters .....	46
Queuing for Arbortext Editor Clients.....	47
Monitoring Queues .....	49

Client requests can be queued, and one or more Queue Managers and queues can be set up to process transactions according to their configuration criteria. When queuing is set up on the server, Arbortext PE Request Manager creates a transaction ID for a request and then passes it to each Queue Manager. When a Queue Manager accepts the transaction, it then offers the transaction to each queue in turn. The queue that accepts the transaction makes the request available to be processed during its active interval, and then it stores the result in the active transaction directory according to its configuration. You can configure how long transactions are kept in the active transaction directory. You can also configure which transactions will be archived, as well as how long they will be kept.

---

## How Queuing Works

Queuing stores transactions for processing at a later time. The queue's configuration criteria determine when its transactions will be processed. Because parameters can interact with each other, it's important to set up a test area while you are configuring your queues.

A request to queue a transaction can come from a variety of clients, such as Arbortext Editor users, custom applications, or web browsers. In each case, the initial response from the Arbortext PE server is to notify the client that the request has been queued and to return the assigned transaction ID. This initial notification can be returned in HTML or XML form.

A Queue Manager is the queuing gatekeeper for incoming requests. A Queue Manager examines a request and determines whether to offer it to queues, and then offers it to any configured Arbortext Publishing Engine queues until one accepts it or they all refuse it. When a queue accepts the request, the request becomes a transaction which is stored in the queue until the Queued Transaction Scheduler can execute it (see [The Queued Transaction Scheduler on page 45](#) for information).

The queue configuration determines how and when queued transactions are available for the Queued Transaction Scheduler to process. Queued transactions are made available during a queue's active time interval. When a transaction is executed, the result is stored in the active transaction directory, and the transaction request is deleted from the queue. Configuration criteria determine this active time interval, order of transaction processing within the queue, simultaneous execution, and so forth.

There is a distinction between a Queue Manager and queues. A Queue Manager controls what gets queued. However, a queue controls how queued transactions are handled, such as their order of processing, when the Queued Transaction Scheduler is allowed to execute them, and other conditions. In distinguishing between Queue Managers and Queues, custom code could determine which requests get queued without having to worry about how queued transactions are stored. Custom code could also control how queued transactions are handled without having to worry about which transactions get queued.

Resource allocation and consumption is managed according to the following:

- An Arbortext PE sub-process pool will only allocate an Arbortext PE sub-process to the Queued Transaction Scheduler if no immediate requests are waiting for an Arbortext PE sub-process from that pool.
- The `maxConcurrentQueuedTransactions` global parameter can set the number of transactions that are allowed to execute simultaneously overall. See [The Global Active Transaction Parameters on page 71](#) for information.
- The `max-concurrent-transactions` queue attribute (set for an individual queue) can set the number of transactions that are allowed to

---

execute simultaneously from that queue. See [The max-concurrent-transactions Attribute on page 96](#) for information.

- The `maxConcurrentQueuedTransactions` Arbortext PE sub-process pool attribute can set the number of Arbortext PE sub-processes from that pool that can be simultaneously allocated to the Queued Transaction Scheduler. Setting this value to zero prohibits queued transactions, and reserves the entire pool only for immediate transactions. See [The maxConcurrentQueuedTransactions Attribute on page 102](#).

When you are ready to set up queuing, the following sections explain the configuration parameters you might use:

- Active transaction parameters described in [The Global Active Transaction Parameters on page 71](#).
- Transaction archive parameters described in [The Global Transaction Archive Parameters on page 74](#)
- Global queuing parameters described in [The Global Queuing Parameters on page 76](#).
- Queue parameters described in [Specifying Queues on page 93](#).

You can also set up Arbortext PE sub-process pools that are dedicated to taking only immediate requests or only queued requests. Refer to [Configuring Sub-process Pools on page 100](#) for information.

---

 **Note**

For backward compatibility, Arbortext PE Request Manager always handles requests from older versions of Arbortext Editor as immediate (rather than queued) requests by default (which requires no special configuration).

---

## Configuring a Queue Manager

A Queue Manager screens requests to determine which requests are offered to the queues, and then it sends responses to the client with information about the queued transaction. Each time Arbortext PE Request Manager receives a request, it determines whether it should pass the request to each configured Queue Manager. Ordinarily, you only need one Queue Manager, but if you have more than one, the request is offered to each, in the order they are defined in `e3config.xml`. A Queue Manager declines a request by returning a null response, in which case the Arbortext PE Request Manager continues to the next defined Queue Manager. If a Queue Manager accepts the request, it returns an HTTP Response to the Arbortext PE Request Manager, which returns the response to the client.

---

A Queue Manager can further filter a request using a `TestSet` parameter to control which requests to examine and whether to offer the request to the queues. See [Specifying Test Sets on page 91](#) for information.

A Queue Manager is a Java object that must implement the `com.arbortext.e3.E3QueueManager` interface. Each `QueueManager` can have an associated list of parameters, as defined by an application that implements the `com.arbortext.e3.E3QueueManager` interface.

## The Arbortext Publishing Engine Queue Manager

One Arbortext `QueueManager` is defined in `e3config.xml`, with its class set to `com.arbortext.e3.QueueManager` and ID `default-qm`. The Arbortext Queue Manager examines each request looking for the HTTP query parameter `queue=yes`, and takes one of the following actions:

- declines to queue a request if the request does not specify the parameter `queue=yes`.
- returns an error if a request includes `queue=yes` but no queue can accept the transaction.
- generates a response page for the client containing the transaction ID if `queue=yes` and a queue accepts the transaction.

The response page will be in either HTML or XML format. The query parameter `response-format` can specify `xml` or `html`. If `response-format=xml` is present, it will return an XML response. If it's not specified, the default is HTML.

## Configuring Queues

A queue is a container of transactions that are awaiting execution. Transactions are placed on a queue by a Queue Manager evaluating incoming requests. A Queue Manager offers the request to each queue in the order they are defined in `e3config.xml`. After a queue accepts it, the transaction waits until the queue's configuration allows the Queued Transaction Scheduler to select it for execution (see [The Queued Transaction Scheduler on page 45](#)). When execution is complete, a queued transaction is deleted from its queue. The transaction is placed in the active transaction directory for the client to retrieve.

Each queue can further filter a request using a `TestSet` parameter to accept or decline a request based on the additional test set criteria. For example, it could filter the request based on whether a Arbortext Editor client requests a PDF (see [Specifying Test Sets on page 91](#) and [Queuing for Arbortext Editor Clients on page 47](#) for information).

A queue is a Java object that implements the interface `com.arbortext.e3.E3Queue`. The sample implementation code for it is found in:

---

`PE_HOME\e3\samples\java\com\arbortext\e3\ArbortextQueue.java`

## The Basic Arbortext Publishing Engine Queue

Arbortext Publishing Engine has one basic Queue implemented in `e3config.xml`. It specifies the class `com.arbortext.e3.ArbortextQueue` with the ID `default-queue`.

It accepts all requests and process them in the order they are received. You can use this queue as a sample to follow for adding and configuring other queues. Queue Managers process queues in the order that they are configured in the `e3config.xml` file. Remember to give each queue a unique name.

The Arbortext queue supports the following basic features:

- Can examine a transaction and either accept it or decline
- Can be enabled or disabled, according to its configuration or manually
- Can be active or inactive, according to its configuration
- Can list the transactions queued within it and determine their priority
- Can allow a transaction to be moved up or down in the list
- Can allow a transaction to be deleted from the queue
- Can hold all transactions in the queue, even when it's enabled and active

Consult [Monitoring Queues on page 49](#) for information on queue administration. See [The Global Queuing Parameters on page 76](#) and [Specifying Queues on page 93](#) for information on configuration parameters for queues.

## The Queued Transaction Scheduler

The Queued Transaction Scheduler is a background process running under the Arbortext PE Request Manager. The scheduler monitors both the queues and Arbortext PE sub-process pools, trying to match queued, active transactions with idle, available Arbortext PE sub-processes. When the Queued Transaction Scheduler is searching for transactions to execute, it will process queues in the order they are configured in `e3config.xml`. By default, it does not require that all transactions on one queue start execution, or complete execution, before it starts executing a transaction on a subsequent queue.

You can set a parameter that consists of a list of queue IDs that delay execution of transactions on a particular queue until the transactions on all of the listed queues are either inactive or empty of all transactions except those being held. See [The previous-queues Attribute on page 96](#) for information.

---

The global parameter

`com.arbortext.e3.transactionArchive.IInterval` controls the interval for checking for queued transactions (see [The Global Queuing Parameters on page 76](#)). It's set to 10seconds by default, and you should not have to change it.

## Queuing Query Parameters

The Arbortext Publishing Engine implementation of a Queue Manager and queuing supports the following HTTP query parameters. These parameters can be used by client applications and web browsers to specify how to handle a queued request.

### Query Parameters for Queuing

Parameter Name	Value	Description
<code>queue</code>	yes or no	Specifies whether to queue a request. If this parameter is omitted or set to <code>no</code> in the request HTTP query, the transaction is treated as an immediate request. The default is <code>no</code> .
<code>queue-priority</code>	1, 2, 3, 4, or 5	Specifies the priority of the request when placing the transaction on a queue. 1 is the highest and 5 is the lowest. <code>queue=yes</code> must also be specified. The default is 3
<code>transaction-name</code>	<i>descriptive-name</i>	Specifies the descriptive text to be used for a transaction name. The specification can include the string <code>\$t</code> , that will be replaced with the unique transaction ID assigned on the Arbortext PE server. For example, <code>transaction-name=Docs_ \$t</code>

## Query Parameters for Queuing (continued)

Parameter Name	Value	Description
response-format	xml or html	Specifies the format of the response that the Queue Manager will return to the client submitting the request. The response will contain the transaction ID. queue=yes must also be specified. The default is html.
notify-email	a valid email address	Specifies an email address to which a notification can be sent when the transaction completes. A notifier must be set up on the Arbortext PE server. Note that this parameter may be specified with both immediate and queued requests.

If you implement a notifier, the notifier needs to receive a `request-parameter` from the HTTP query set to the value of the query parameter name that will be used to submit an email address. To use the built-in `notify-email` query parameter, the `request-parameter` must be set to `notify-email`. When the client submits a request, `notify-email=`  
`"user@mycompany.com"` must also be part of the query request. See [Configuring a Notifier on page 97](#) for information about the sample package, including an example for using it.

## Queuing for Arbortext Editor Clients

Arbortext Editor users can be permitted to queue their publishing requests. They can then retrieve the resulting output using **Tools ▶ Queued Transactions**.

Publishing requests sent from Arbortext Editor will include the `ati-operation-type` parameter and the publishing type as its value. The values for this parameter are used to determine what is displayed as the **Operation** on the **Queued Transaction List** and **Completed Transaction List** web pages on the Arbortext PE server, as well as the **Transaction Archive** web page. Refer to [Monitoring Queues on page 49](#) and [Requesting the Queue Reports on page 118](#) for more information.

Queuing can be set up for Arbortext Editor users by configuring its **Publishing Engine** preferences or choosing queuing from a publishing dialog box. Refer to [Using Arbortext Publishing Engine for Publishing Documents help topic](#) in the Arbortext Help Center.

The `ati-operation-type` parameter can specify the following:

- `appsave`

- 
- means **Generate Application Save**
  - epub
    - means **Publish For EPUB**
  - htmlfile
    - means **Publish HTML**
  - htmlhelp
    - means **Publish HTML Help**
  - import
    - means **Import Document**
  - lnr
    - means **Generate Line Numbers**
  - pdf
    - means **Publish PDF**
  - rtf
    - means **Export to RTF**
  - web
    - means **Publish Web**
  - xsl
    - means **Publish Using XSL**

You can use the `ati-operation-type` parameter to configure a queue to accept transactions of a specific publishing operation by creating a `TestSet` that looks for the appropriate publishing types. See [Specifying Test Sets on page 91](#) for information.

When you submit queued transactions to the server, you can assign a descriptive name to the transaction. Transactions are identified by their unique transaction ID, so a queued transaction name allows you to provide a more readable description for the published document. Given that you can specify any text for a queued transaction name, it is not unique for finding a particular transaction or retrieving specific results. You can also set the value for individual publishing requests in the Transaction Name field in the File ▶ Publish dialog boxes when publishing a document.



---

## Monitoring Queues

Several tools are available to monitor queues and their transactions. The **Queue List** link on the Arbortext Publishing Engine index page (described in [Monitoring and Reporting Using a Web Browser on page 21](#)) displays the list of configured queues.

From the **Queue List** web page, you can get information about the queues:

- **Info** link displays the **Queue** page containing the queue's configuration.
- **Enabled** reports the queue state.
- **Action** lets you toggle the enabled/disabled state.
- **Active** reports whether the queue is currently active.
- **Queued Transactions** displays the number of queued transactions. The numeral link displays the **Queued Transaction List** page.
- **Completed Transactions** displays the number of completed transactions, The numeral link displays the **Completed Transaction List** page.

For more information on queuing reports, refer to [Requesting the Queue Reports on page 118](#). For more information on transaction list reports, refer to [The Transaction List Page on page 119](#).



# 5

## Understanding Publishing Rules

Managing Publishing Rules.....	54
Deploying Publishing Rules.....	54

A publishing rule is a set of parameters that are applied to a document publishing process. Some parameters are general and apply to all outputs, such as a stylesheet or profiling. Other parameters are specific to the output type, such as HTML encoding, PDF configuration file, or Web frameset. A publishing rule offers a way to save favorite publishing choices to be reused. Multiple rules can be grouped together into a rule set, and then used to publish a document multiple times to multiple outputs.

Arbortext Editor users can create a publishing rule on the fly and save it to use again when publishing the next time. A Arbortext Editor user defines a publishing rule by choosing:

- **Tools ► Administrative Tools ► Publishing Rules**
- **Create Rule** from any of the dialog boxes that are launched from the choices on the **File ► Publish** menu:
  - **For Web**
  - **For HTML Help**
  - **For EPUB**
  - **HTML File**
  - **PDF File**
  - **RTF File**
  - **Using XSL**

---

 **Note**

**Import/Export, Print Preview, and Print** do not support publishing rules.

---

Arbortext Editor users can then choose **File ▶ Publish ▶ Using Rule** and select a rule or rule set from the list to publish the document. Publishing rules are documented in the online help for Arbortext Editor. Refer to **Publishing Rules Overview** in Arbortext Help Center, under **Authoring with Arbortext Editor ▶ Help ▶ Printing and Publishing ▶ Publishing Rules**

Publishing rules can also be specified by client programs or by the built-in Arbortext Publishing Engine `f=convert` function parameter. For more information, refer to the *Programmer's Guide to Arbortext Publishing Engine*. Publishing rules parameters are documented in the *Customizer's Guide*.

The publishing process in effect at the time the publishing rule is saved determines the type of publishing rule. There are two types of publishing rules as determined by the publishing mode the user is currently using:

- A `local` publishing rule is created if Arbortext Editor is using Arbortext Styler or a local Print Composer option license for publishing. A `local` publishing rule can be saved in the following locations:
  - `publishingrules` subdirectory of the application and custom directories
  - `publishingrules` directory of the Arbortext Editor installation
  - the user's home directory

`local` publishing rules can be selected for publishing only if Arbortext Editor is using Arbortext Styler or a local Print Composer option license.
- A `server` publishing rule is created if Arbortext Editor is using Arbortext Publishing Engine for publishing (**Tools ▶ Preferences ▶ Publishing Engine**, the preference for **Use Publishing Engine** is checked). In this case, a `server` rule is still saved locally (in the same possible locations as `local` publishing rules).

`server` publishing rules can be selected for publishing only if Arbortext Editor is using Arbortext Publishing Engine for publishing (meaning the **Use Publishing Engine** preference is checked).

Arbortext Editor looks for publishing rule files in several locations, and offers the user a list of publishing rules that are of the type supported by their current publishing mode:

- 
- in a `publishingrules` subdirectory of application and custom directories
  - in the `publishingrules` directory of the Arbortext Editor installation
  - in the Arbortext Editor user's home directory
  - on the Arbortext PE server, if Arbortext Editor is using Arbortext Publishing Engine for publishing.

Arbortext Editor users can check the online help for more information on creating publishing rules and rule sets.

---

## Managing Publishing Rules

An administrator can manage a set of reusable publishing parameters by adopting a strategy for creating, saving and deploying publishing rules. Because the type of rule is determined by the publishing process in effect when the rule is created, you need to be aware of the following:

- If the user is using Arbortext Styler or a local Print Composer option license for publishing, the rule is saved as a `local` rule. A `local` rule is available to the Arbortext Editor user only when publishing using one of these options. The user can't select a `server` publishing rule.
- If the user sends publishing requests to Arbortext Publishing Engine (the **Publishing Engine** preference for **Use Publishing Engine** is checked), the rule is saved as a `server` rule. A `server` rule is still saved locally. A `server` rule is available to the Arbortext Editor user only when publishing with Arbortext Publishing Engine. The user can't select a `local` publishing rule.

A Arbortext Editor client can display a `server` publishing rule list consisting of some combination of `server` rules that were created and saved locally combined with `server` rules deployed on the Arbortext PE server. The next section describes deployment strategies.

The Publishing Configuration report contains information about publishing rule files. It includes only those publishing rule files that are free of errors. If a rule file is incorrectly formed, has the same unique ID as another rule file, or some other error, that information will be noted in the publishing configuration log, but it will not be included in the Publishing Configuration report. Rule files with errors are not presented as choices to Arbortext Editor clients. Refer to [Publishing Configuration on page 26](#) for information.

For information on customization of Publishing Rules using advanced parameters, consult *Customizing Publishing Rules* in the *Customizer's Guide*.

## Deploying Publishing Rules

A publishing rule or rule set intended for deployment on the Arbortext PE server must be created using Arbortext Publishing Engine Interactive. This means that when the rule file is saved, its type will be `local` on the server, but it will be offered to Arbortext Editor clients and other client applications as a `server` rule. Publishing rules can be put in the server's `custom\publishingrules` directory or a `custom\publishingrules` directory defined by the `APTCUSTOM` environment variable.

An administrator can also deploy `local` publishing rules for multiple Arbortext Editor users who are not using Arbortext Publishing Engine (meaning they have Arbortext Styler or a local Print Composer option license ). The rule files can be

---

made available from a common `custom\publishingrules` directory accessible to their local machines, where the location is most likely defined by setting the path in the user's *APTCUSTOM* environment variable.





# 6

## Windows Configuration

Configuring a User Account on Windows .....	58
Using the Arbortext Publishing Engine Configuration Program .....	60
Integrating Arbortext Publishing Engine with Apache Tomcat .....	66

This chapter provides information on setting up a user account for Arbortext Publishing Engine, optional configuration performed by the Arbortext Publishing Engine Configuration program, and integrating Arbortext Publishing Engine with Tomcat.

---

## Configuring a User Account on Windows

By default, Arbortext PE sub-processes run under a special user account called *SYSTEM* that has restricted access. You may want to configure Arbortext Publishing Engine to run under a specified user account.

The default Arbortext Publishing Engine user account doesn't have network privileges, so it can't access any network resources such as a file shared on another host or a remote printer. For example, if you were to write a custom Arbortext Publishing Engine application that sends a print job to a network printer, you would need to configure Arbortext Publishing Engine to run under a specified user account.

If Arbortext Publishing Engine needs to specify a file on a network using a UNC path (uniform naming convention, which takes the form `\\servername\sharedir`), the default Arbortext Publishing Engine *SYSTEM* account wouldn't be able to access it. However, a user account can be configured to access files specified by a UNC if the user account has the proper permissions and the UNC file system shares the directories with the Arbortext Publishing Engine user account.

If Arbortext Publishing Engine is configured to run as a specified user account, you must log in using that account if you want to run Arbortext Publishing Engine Interactive. When Arbortext Publishing Engine is configured this way, Windows doesn't allow Arbortext Publishing Engine Interactive to be run under a different user account for security reasons. However, any Administrator level user account can run the **Arbortext Publishing Engine Configuration** program.

When you are configuring Arbortext Publishing Engine to run as a specific user account on Windows, the Arbortext Publishing Engine user account:

- can be set up to access network resources, if needed.
- should have internet access as it handles HTTP traffic.
- should have an interactive login so that someone can log in to perform testing and troubleshooting.
- should have a password that doesn't expire so that Arbortext Publishing Engine won't suddenly terminate if the account is locked without warning. Of course, passwords should still be changed regularly.
- should have a default printer selected in the Windows **Printers** window. The default printer must be a PostScript printer if you will be producing PostScript.
- does not need to have local administrator privileges on the server that it's installed on.

---

## To set up Arbortext Publishing Engine as a specific user account:

You'll need to set permissions and restrictions for a specified Arbortext Publishing Engine user account. The system administrator familiar with setting up user accounts, especially your site's security policies and user privileges, may need to perform the following steps.

---

### Note

Your site may have security policies in place that require an IT administrator familiar with those policies to perform these steps.

---

1. You should be logged in with Administrator privileges on the host machine where Arbortext Publishing Engine is installed.
2. In **Administrative Tools**, open **Computer Management**. In the **Computer Management** window, find the `Local Users and Groups` folder and display the `Users` folder under it. From the **Action** menu of the **Computer Management** window, choose **New User**. In the **New User** window, create a new account and its password. Set the properties you want for the password and click **Create**.
3. In **Administrative Tools**, open **Local Security Policy**. In the **Security Settings** window, display the `Local Policies` folder under it. Click the `User Rights Assignment` subdirectory to display a list of possible settings. Find **Log on as a batch job** in the **Policy** list and double click on it.
4. In the **Log on as a batch job Properties** dialog box, click the **Add User or Group** button. In the **Select Users or Groups** dialog box, be sure the object type includes **Users** and the **Location** is the local server machine. Enter the Arbortext Publishing Engine user account name in the object name text box. Click the **OK** button to return to **Log on as a batch job Properties**. Click **OK** to save changes and close the dialog box. Set any other privileges you want for this account.
5. In **Administrative Tools**, open **Component Services** and navigate to **Computers > My Computer > DCOM Config**. Find and display the **Properties** for the **Arbortext Editor** entry by right clicking on the entry.
6. In the **Arbortext Editor Properties** dialog box, click the **Identity** tab. Choose the **This user** option. Fill in the **User** and **Password** fields for the Arbortext Publishing Engine account you created. You need to specify the domain or computer name as well as the user name; enter the account name and then click the **Browse** button to specify the **User** information.
7. Click **OK** to accept the changes to **Arbortext Editor Properties**. Click **OK** again to exit.

- 
8. You may need to check that the Arbortext Publishing Engine user account has permission to write to some specific directories. For example, you will want to be sure it can write to temporary or log directories of the Windows system and other applications, such as the servlet container directories that store log files. For instance, you would want to check permissions for the following:
    - The temporary directory set by the System's environment variable.
    - Java log and temporary directories for the servlet host application, for example, Tomcat's `logs` and `temp` directories. On the Arbortext Publishing Engine index page, click the **Java Properties** link to retrieve the JVM System Properties page. The `java.io.tmpdir` property value will display the location.
    - The `C:\ProgramData` directory.
    - Check any file path specified in the `e3config.xml` file, such as the transaction archive directory.

Right click on the appropriate directory and choose **Properties**. Choose **Security** and check if the Arbortext Publishing Engine user account is in the list and has Full Control. If not, you will need to add the Arbortext Publishing Engine user account and give it **Full Control** for the folder.

9. Restart the system. The next time Arbortext Publishing Engine runs, it will run as the Arbortext Publishing Engine user account you created.

## Verifying the Specified User Account

You can verify that Arbortext Publishing Engine is running under the Arbortext Publishing Engine user account. The **Arbortext Diagnostics** utility is a program available on Windows that displays tracing information. You can launch it from the **Arbortext Diagnostics** shortcut in your PTC program group. The **Arbortext Diagnostics** window contains the **Module List** tab that displays the Arbortext PE sub-process information and the **Trace Log** tab that displays tracing output. The **Module List** panel should display the Arbortext Publishing Engine user account name instead of `SYSTEM`.

## Using the Arbortext Publishing Engine Configuration Program

The **Arbortext Publishing Engine Configuration** program lets you choose a default printer, integrate Arbortext Publishing Engine with Tomcat, and generate a diagnostic report. You can click a button that opens the `e3config.xml` file to edit other Arbortext Publishing Engine configuration settings.

---

The **Apply** button appearing on each tab of **Arbortext Publishing Engine Configuration** applies your changes as specified. If you're using the Tomcat servlet container, it automatically stops and restarts Tomcat for the changes to take effect. You can continue to make further configuration changes without closing and reopening the **Arbortext Publishing Engine Configuration** program each time.

Each tab in the **Arbortext Publishing Engine Configuration** has its own **Item Help** that explains the values you can set in each of the fields. Select a field to display its **Item Help**.

---

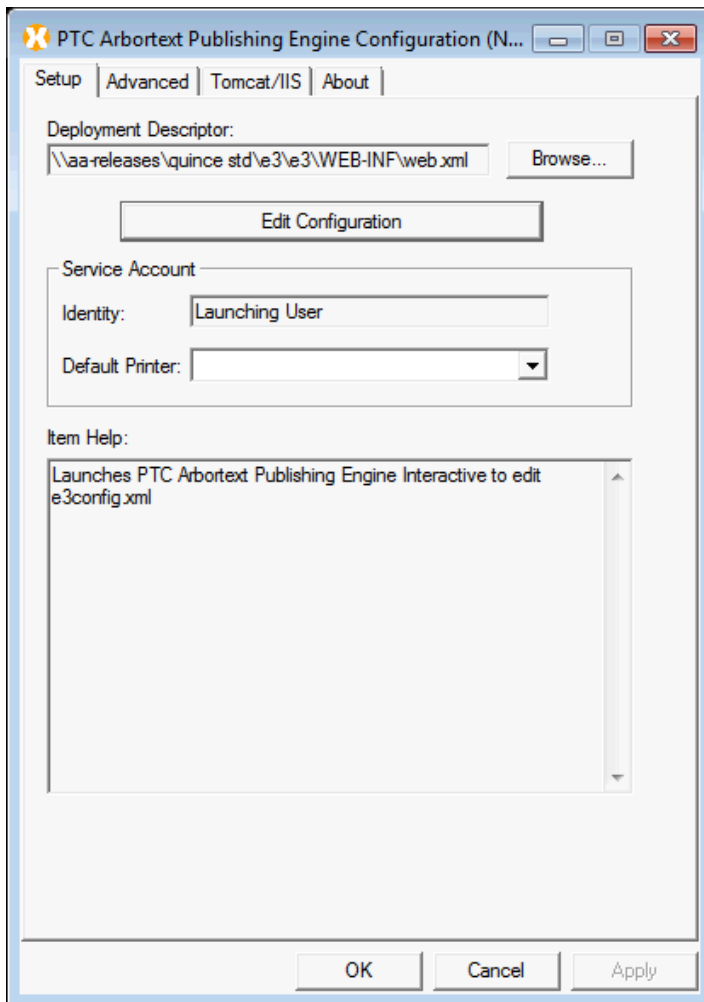
 **Note**

Do not use a mapped or `subst` drive to browse for your Arbortext Publishing Engine installation to run **Arbortext Publishing Engine Configuration**. When Arbortext Publishing Engine runs as the SYSTEM account, a mapped drive will not be available to it, which then causes problems for the COM server.

---

## Setup Tab

The **Setup** tab provides some basic information about the setup of Arbortext Publishing Engine.



- The location of the Arbortext Publishing Engine `web.xml` is displayed in the **Deployment Descriptor** field.
- **Edit Configuration** launches Arbortext Publishing Engine Interactive and automatically opens the `e3config.xml` file containing the Arbortext Publishing Engine configuration settings.
- **Service Account** displays the **Identity** of the Arbortext Publishing Engine. If the value in this field is `Launching User`, then the Arbortext PE subprocesses run under the servlet container account. Most servlet containers run as a Windows service under the Windows `SYSTEM` account. For more information about setting up the Arbortext Publishing Engine as a specific user account, refer to *Installation Guide for Arbortext Publishing Engine*.
- **Service Account** group displays a **Default Printer** for the Arbortext Publishing Engine user account. If you use Arbortext Publishing Engine to produce PostScript output, you need to choose a PostScript printer from the list. You don't need to choose a PostScript printer for producing PDF.

---

If the Arbortext Publishing Engine is running as a specified user, you need to choose the default printer for the Arbortext Publishing Engine user account in the Windows **Printers** window.

---

 **Note**

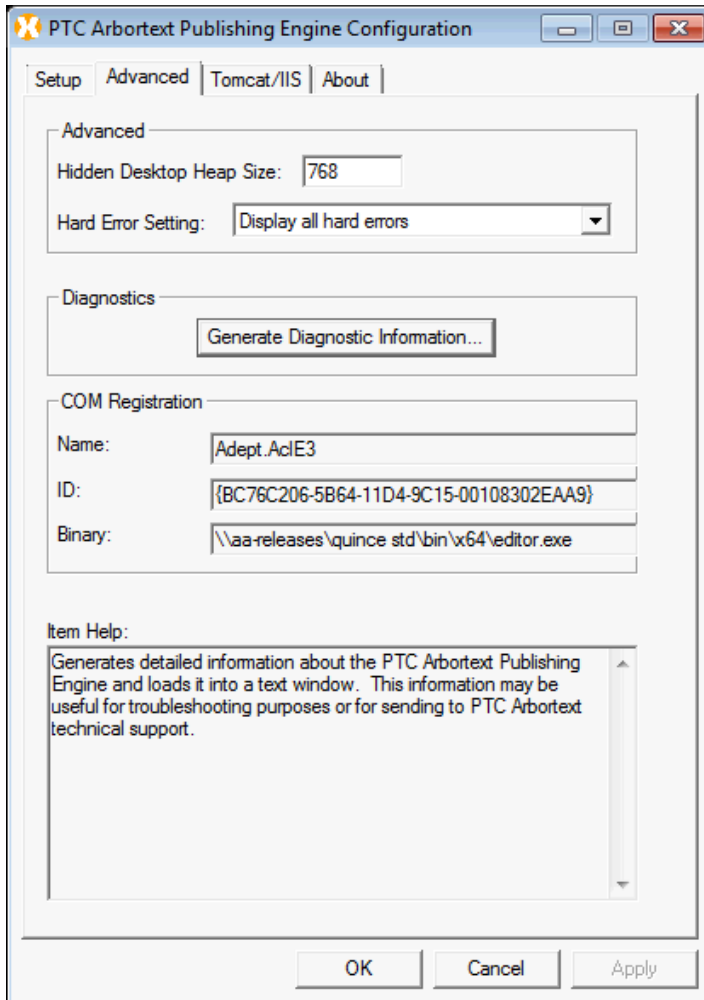
If a print request from Arbortext Editor uses APP, a PDF will be returned to the user for printing.

---

To set the default printer for a specified Arbortext Publishing Engine user account, you must login under this account and set the default printer as you would for any Windows account. However, the **Arbortext Publishing Engine Configuration** doesn't display the default printer even after you have manually set it. The **(must set manually)** message persists in the **Default Printer** field. You can try a test using PostScript output to confirm the printer is recognized by the Arbortext Publishing Engine.

## Advanced Tab

The **Advanced** tab displays several group boxes for managing Arbortext Publishing Engine advanced configuration settings.



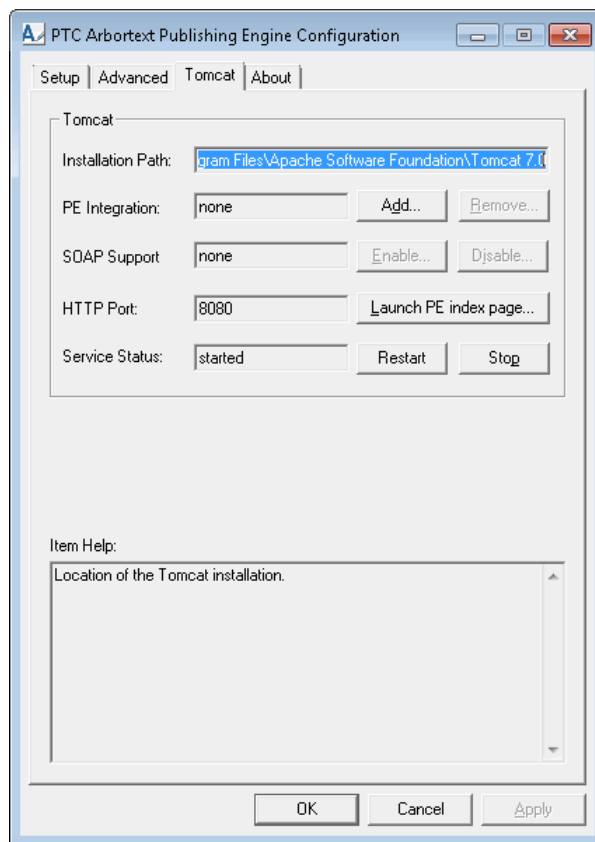
- The **Advanced** group box settings control Arbortext Publishing Engine behavior in managing aspects of its environment.
  - The `Hidden Desktop Heap Size` sets the system heap size for the Arbortext PE sub-processes. The Arbortext Publishing Engine automatically suggests an appropriate value based on the number of Arbortext PE sub-processes specified in the **Setup** tab.
  - `Hard Error Setting` determines whether the hard error popup window is displayed when an application fails. During the Arbortext Publishing Engine development phase at your site, you should keep this setting at the default `Display all hard errors` for troubleshooting purposes. In a production environment, this setting should be changed to `Suppress all hard errors` so that the Arbortext PE Request Manager can automatically restart an Arbortext PE sub-process without waiting for a user at the keyboard to dismiss the hard error popup window. All errors are



reported to the Windows Event log whether they are displayed or suppressed, so errors can be checked there.

- The **Generate Diagnostic Information** button produces a detailed report about Arbortext Publishing Engine and its environment to assist with troubleshooting. You should include the diagnostic report with any request for technical product support from Arbortext.
- The **COM Registration** box verifies the Arbortext PE sub-process binary that registered with the COM server on the system. This information might be useful for troubleshooting.

## Tomcat Tab



The **Tomcat** tab has a **Tomcat** group box which displays the Tomcat installation path, whether SOAP is enabled, the HTTP port Tomcat monitors, and whether the Tomcat service is stopped or started. You can add or remove the automatic integration between this installation of Arbortext Publishing Engine and Tomcat.

When you click **Add**, the **Arbortext Publishing Engine Configuration** program will integrate with Tomcat. You can enable SOAP at the same time by choosing **Enable**. For information on using a SOAP client with Arbortext Publishing Engine, consult the *Programmer's Guide to Arbortext Publishing Engine*.

---

 **Caution**

There are potential security issues with SOAP and the Axis2 libraries. PTC recommends that you use HTTPS if you use SOAP. For more information, see *Arbortext Publishing Engine Security Framework* in the *Configuration Guide for Arbortext Publishing Engine*.

---

The **Launch PE index page** button tests Arbortext Publishing Engine configuration by loading the Arbortext Publishing Engine index page through the Tomcat servlet container.

Consult [Integrating Arbortext Publishing Engine with Apache Tomcat](#) on page 66 for more information on performing the integration.

---

 **Note**

- Tomcat has an AJP connector defined in `server.xml` which monitors port 8009. However, if port 8009 is already in use, Tomcat doesn't address the conflict. Arbortext Publishing Engine Configuration configures `isapi_redirect.properties` to use port 8009 to integrate with Tomcat.
- 

## About Tab

The **About** tab provides the version and copyright information for the **Arbortext Publishing Engine Configuration** program.

## Integrating Arbortext Publishing Engine with Apache Tomcat

If the **Arbortext Publishing Engine Configuration** program detects Tomcat on the system, it prompts you during installation to integrate Tomcat with Arbortext Publishing Engine. Integrating with Tomcat is required for Arbortext Publishing Engine. If you choose to bypass these prompts, you can still perform these integrations later.

---

 **Note**

You must ensure that Tomcat is configured in line with current security best practices.

---

---

## To integrate Arbortext Publishing Engine with Tomcat

1. If it's not already open, start the **Arbortext Publishing Engine Configuration** program. The **Arbortext Publishing Engine Configuration** program is available from its Arbortext Publishing Engine program group shortcut. Choose the **Tomcat** tab.
2. In the **Tomcat** group box, look at the value in the **PE Integration** field. If the value is *none*, click the **Add** button to add Arbortext Publishing Engine information to the Tomcat configuration file `TOMCAT_HOME\conf\server.xml`. If the field value is *other*, you may need to investigate another Arbortext Publishing Engine installation before you can proceed. If the value is *other*, choosing **Add** replaces the other Arbortext Publishing Engine integration with this new one.
3. If the value in the **Service Status** field is *stopped*, you can click the **Restart** button. You don't need to **Stop** the service before making changes; the **Arbortext Publishing Engine Configuration** program will stop and restart the Tomcat service when you choose **OK** to exit.
4. Click the **Launch PE index page** button to open the Arbortext Publishing Engine index page. The Arbortext Publishing Engine page has links for getting information, converting a demo document to a variety of output formats, running test applications, and obtaining the list of Arbortext Publishing Engine web services. If the Arbortext Publishing Engine index page appears in a web browser, Tomcat is running, and it has been successfully configured to find the Arbortext PE Request Manager.
5. On the Arbortext Publishing Engine index page, you can click on the `Status` link to test whether Arbortext PE Request Manager can return a status report. If a status report is returned, then Tomcat is successfully running the Arbortext PE Request Manager.
6. To test that Arbortext Publishing Engine can handle a request to do work, click on one of the links under **Test Arbortext Publishing Engine**. If information or the requested converted file is returned, then Arbortext Publishing Engine is working properly.



## Setting Configuration Parameters

The e3config.xml Configuration File .....	71
Arbortext Publishing Engine Global Parameters.....	71
Specifying a Request Handler .....	83
Specifying Request Selectors.....	89
Specifying Test Sets .....	91
Specifying Caches.....	92
Specifying Queues .....	93
Configuring Sub-process Pools .....	100
Specifying the AllowedFunctions List .....	107
Specifying Initializers .....	109

The configuration file `e3config.xml` contains parameter definitions used as configuration settings for Arbortext Publishing Engine. The default values are adequate for most sites to begin development and testing. As you build your Arbortext Publishing Engine production system, you'll need to review and possibly enable or change parameter or attribute values in this file.

The Arbortext Publishing Engine Java, JavaScript, VBScript, and ACL sample programs retrieve the Arbortext Publishing Engine configuration parameters and values through the Arbortext Publishing Engine HTML index page. These sample programs are explained in [Monitoring and Reporting Using a Web Browser on page 21](#).

You can also retrieve these configuration parameters from your custom applications:

- For Java or JavaScript applications, call the **E3ApplicationConfig** interface.
- For VBScript or ACL applications, call the **PEAppConfig** package.

Consult the *Programmer's Guide to Arbortext Publishing Engine* for information on using the Arbortext Publishing Engine interfaces and packages when writing custom applications.

---

To become more familiar with the specific parameters and attributes used in this file, refer to the remaining sections in this chapter. After you've reviewed their explanations, you can proceed to edit the file to make changes appropriate for your site.

---

## The e3config.xml Configuration File

You will need to review and edit the `e3config.xml` configuration file, which is an XML file that Arbortext Publishing Engine reads each time it starts.

If you are interested in reviewing the document type and DCF files for `e3config.xml`, they're located in `PE_HOME\doctypes\e3config`.

### Editing the e3config.xml file

1. Open the file in Arbortext Publishing Engine Interactive on the server where you've installed Arbortext Publishing Engine. You can start Arbortext Publishing Engine Interactive from its shortcut on your Arbortext program group.
2. Choose **File** ► **Open** and locate the file:  
`PE_HOME\e3\e3\WEB-INF\e3config.xml`
3. Customize existing parameters and attributes or add new ones as needed. You will need to be familiar with your site's Arbortext Publishing Engine implementation and any custom applications.
4. You can choose a parameter element and then choose **Edit** ► **Modify Attributes** to display the **Modify Attributes** dialog box. You'll supply a parameter name and a parameter value.
5. Choose **File** ► **Save** to save your changes.

## Arbortext Publishing Engine Global Parameters

There are several global parameters that can be configured at the beginning of the `e3config.xml` configuration file. These parameters set temporary directory locations, debugging levels, transaction behavior, archiving behavior, and queuing behavior. These global parameters are explained in the following sections.

### The Global Active Transaction Parameters

The following global parameters manage active transaction storage before, during, and after processing.

- `com.arbortext.e3.transactionDirectory`  
Specifies the directory where Arbortext Publishing Engine puts transaction subdirectories. For every request, Arbortext Publishing Engine will create a subdirectory in this directory and store intermediate files in it, as well as build the response to the request. The default directory is explicitly set to `activeTransactions`. Its location is in the directory specified by

---

`com.arbortext.e3.tempFileDirectory`, described in [The Global `com.arbortext.e3.tempFileDirectory` Parameter on page 82](#).

If your implementation will process a large number or size of transactions, you should put the directory in a location that is not in the server's temporary storage area. You should also periodically monitor the completed transactions directory to assess the disk space being used.

- `com.arbortext.e3.transaction.maxCompletedTransactionAge`

This parameter specifies the maximum time, in hours, that a transaction directory will be kept in the Active Transaction Directory. This parameter applies only to queued transactions that have been completed but the results have not been retrieved. The default is explicitly set to 168 hours (or one week).

When Arbortext Publishing Engine finishes processing a queued transaction, it notes the time. When the interval specified by this parameter has elapsed, Arbortext Publishing Engine may copy the transaction directory into the transaction archive (according to the setting of the `com.arbortext.e3.transactionArchive.selector` parameter). Either way, the transaction directory is then deleted.

Queued transaction results that have been retrieved are deleted according to the interval specified by `maxRetrievedTransactionAge`.

- `com.arbortext.e3.transaction.maxRetrievedTransactionAge`

This parameter specifies the maximum time, in hours, that a transaction directory will be kept in the Active Transaction Directory. This parameter applies only to queued transactions that have been completed, and the results have been retrieved at least once. The default is explicitly set to 48 hours (two days).

After Arbortext Publishing Engine finishes processing a queued transaction, it's available for retrieval. When Arbortext Publishing Engine receives and processes the first request to retrieve a queued transaction result, it notes the time. When the interval specified by this parameter has elapsed, Arbortext Publishing Engine may copy the transaction directory into the transaction archive (according to the `com.arbortext.e3.transactionArchive.selector` parameter). Either way, the transaction directory is then deleted.

Queued transaction results that have not been retrieved are deleted according to the interval specified by `maxCompletedTransactionAge`.



---

## Note

If a transaction directory was created for an immediate (non-queued) request, Arbortext Publishing Engine will copy the transaction directory into the transaction archive if appropriate and delete it from the active transaction directory immediately upon completion.

---

## The Global Transaction Name Parameter

The `com.arbortext.e3.defaultTransactionName` parameter specifies the descriptive name to use for all incoming requests that do not already specify a transaction name. The specification can include the string `$t`, to specify the unique transaction ID assigned on the Arbortext PE server as part of the transaction name. For example:

```
<Parameter name="com.arbortext.e3.defaultTransactionName"
  value="unnamed-$t" />
```

The default value is explicitly set to `tran-$t`.

Arbortext Editor clients can specify a transaction name using the **Queued Transaction Names** dialog box (available from **Publishing Engine** category of **Tools ▶ Preferences**) or the **Transaction Name** field on the **File ▶ Publish** set of dialog boxes.

Applications can specify the query parameter `transaction-name`. Refer to [Queuing Query Parameters on page 46](#) for information.

## The Arbortext Publishing Engine Security Framework Parameter

The `com.arbortext.e3.enableSecurityFramework` parameter specifies whether the Arbortext Publishing Engine security framework is enabled or disabled. For example:

```
<Parameter name="com.arbortext.e3.enableSecurityFramework"
  value="true" />
```

The default value is `false`.

The security framework allows every request sent to the Arbortext PE Request Manager to be classified as "disabled", "unrestricted", or "restricted" based on users and groups defined in Apache Tomcat. Refer to [Arbortext Publishing Engine Security Framework on page 27](#) for framework capabilities and details on configuring the framework. If the framework is disabled, none of the described request processing takes place and the Arbortext PE Request Manager not authenticate requests.

---

 **Note**

While enabling the Arbortext Publishing Engine security framework provides a layer of security against improper access to Arbortext Publishing Engine, it should be considered as only one component of your site's broader security plan.

---

## The Global Transaction Archive Parameters

The following global parameters manage the transaction archive capability, described in [Monitoring the Transaction Archive on page 115](#). Use **Transaction Archive** link on the Arbortext Publishing Engine index page to view the archived transactions list, described in [Monitoring and Reporting Using a Web Browser on page 21](#), .

- `com.arbortext.e3.transactionArchiveDirectory`  
Specifies the directory where archive entries are stored. The default is explicitly set to the subdirectory `transactionArchive`, under the temporary directory specified by `com.arbortext.e3.tempFileDirectory`, described in [The Global `com.arbortext.e3.tempFileDirectory` Parameter on page 82](#).  
If your implementation will archive a large number or size of transactions, you should put the directory in a location that is not in the server's temporary storage area. You should also periodically monitor the transactions held in the archive; see [Monitoring the Transaction Archive on page 115](#).
- `com.arbortext.e3.alternateTransactionArchiveDirectory`  
Specifies the alternate directory where secure archive entries are stored. The default is explicitly set to the subdirectory `alternateTransactionArchive`. You can supply an absolute path to the location, or a relative path under the temporary directory specified by `com.arbortext.e3.tempFileDirectory`.
- `com.arbortext.e3.transactionArchive.enable`  
Specifies whether the transaction archive is enabled. By default, it's set to `true`. Specify `false` to turn off archiving transactions.  
If it is set to `true`, then a transaction can be archived, as determined by the `com.arbortext.e3.transactionArchive.selector` (set to any valid value other than `none`). The values of the `test-archive-`

---

`transaction` or `test-alternate-transaction-archive` test sets determine where the transaction should be archived.

- `com.arbortext.e3.transactionArchive.clearOnStart`

Specifies whether to clear the transaction archive when Arbortext Publishing Engine starts.

  - `false` (explicitly set as the default) keeps archive entries when Arbortext Publishing Engine starts.

Keeping archived entries means the archive keeps all its entries each time Arbortext Publishing Engine starts.
  - `true` clears archive entries each time Arbortext Publishing Engine starts.
- `com.arbortext.e3.transactionArchive.maxAge`

Specifies the maximum time in hours that an entry will remain in an archive before it is automatically deleted. A value of 0 means never delete archive entries. The default is explicitly set to 48 hours.
- `com.arbortext.e3.transactionArchive.maxSize`

Specifies the maximum size in megabytes that a transaction archive may occupy on disk. If an entry causes the archive to exceed this size, entries will be deleted starting with oldest first, until the archive is less than the maximum size. A value of 0 means no deletions will be performed based upon maximum size. The default is explicitly set to 500.
- `com.arbortext.e3.transactionArchive.selector`

Specifies the filters for determining which requests are saved in the archive, as determined by the test sets for `test-archive-transaction` and `test-alternate-transaction-archive` (see [Specifying Test Sets on page 91](#) for more information).

  - `none` saves no entries.
  - `error` saves error entries, including requests with a response other than 200, requests which cause an Arbortext PE sub-process to terminate abnormally, and requests for which transmission of the response to the client fails.
  - `log` (explicitly set as the default) saves error entries, as well as requests with log entries or intermediate files.
  - `all` saves all requests, including successful requests.
- `com.arbortext.e3.transactionArchive.testSet`

Specifies the name of the test set that identifies transactions that should be archived. This parameter is set to `archive-transaction-test`, the test set ID as provided by default in `e3config.xml` (see [Specifying Test Sets on](#)

---

[page 91](#) for more information). The

`com.arbortext.e3.transactionArchive.selector` parameter determines whether archiving is required.

This parameter archives transactions even if

`com.arbortext.e3.transactionArchive.enable` is set to `false`.

- `com.arbortext.e3.transactionArchive.alternateLocation.testSet`

Specifies the name of the test set that identifies transactions that should be archived in an alternate location. This parameter is set to `archive-transaction-to-alternate-location-test`, the test set ID as provided by default in `e3config.xml` (see [Specifying Test Sets on page 91](#) for more information). The

`com.arbortext.e3.transactionArchive.selector` parameter determines whether archiving is required.

This parameter archives transactions even if

`com.arbortext.e3.transactionArchive.enable` is set to `false`.

- `com.arbortext.e3.transactionArchive.threadInterval`

Specifies the number of seconds between execution of the transaction archive management thread. The thread checks for transactions completed since the last run, as well as for transactions older than the interval specified by `com.arbortext.e3.transactionArchive.maxAge`. The default is explicitly set to 10.

The thread will also check for completed transactions that should be archived, as well as archived completed requests that should be deleted, as configured by the `maxCompletedTransactionAge` and `maxRetrievedTransactionAge` parameters.

## The Global Queuing Parameters

The following global parameters set the server queuing policies that will be used by Arbortext Publishing Engine clients.

- `com.arbortext.e3.maxConcurrentQueuedTransactions`

Specifies the maximum number of queued transactions that may run concurrently.

The default is set to `-1`, which means there is no limit, and the Queued Transaction Scheduler will execute as many queued transactions as it can match with idle Arbortext PE sub-processes.

---

You can set to 0 to prevent any queued transactions from executing. Transactions may still be queued, but they will never be executed.

- `com.arbortext.e3.scheduler.threadInterval`

Specifies the number of seconds between execution of the Queued Transaction Scheduler. The thread checks for queued transactions ready for processing and idle Arbortext PE sub-processes, subject to the value specified for the global `maxConcurrentQueuedTransactions` parameter and the `maxConcurrentQueuedTransactions` parameter specified for each Arbortext PE sub-process pool. The default is 10 seconds. You should not need to change this value.

Be sure you refer to [Specifying Queues on page 93](#) for more information on the queue configuration parameters.

## Global Queuing Parameters for Arbortext Editor Clients

In addition, the following parameters apply specifically to a site where Arbortext Editor clients are using Arbortext PE server to fulfill publishing requests. These parameters control whether the Arbortext Editor client is allowed to queue a request and to, optionally, submit information to receive notification about the completed transaction.

The settings for these parameters are included in the Publishing Configuration report that is sent to Arbortext Editor clients using Arbortext Publishing Engine for publishing. The Publishing Configuration document establishes the queuing policies for Arbortext Editor clients. Refer to [Requesting a Publishing Configuration Report on page 121](#) for information on looking at this report on the server.

- `com.arbortext.e3.queueCompositionOperations`

Specifies whether Arbortext Editor clients are allowed to submit queued publishing requests. The possible values are `always`, `optional`, and `never`. By default, this value is explicitly set to `optional`, which allows the Arbortext Editor user to choose whether to queue a request. The value `never` means Arbortext Editor clients will never be permitted to queue a request..

---

### Note

For backward compatibility, Arbortext PE Request Manager always fulfills requests from 5.3 and older versions of Arbortext Editor immediately by default (which requires no special configuration).

---

- `com.arbortext.e3.compositionIdentificationPolicy`

---

Specifies a valid HTTP query parameter name used to identify the user making a publishing request. The default is explicitly set to `header`.

When set to `header`, Arbortext Editor publishing requests set the HTTP `From` header as the query parameter name that will identify the user. The value is then set to the user ID under which Arbortext Editor is running (displayed in **Identification** in **Tools** ▶ **Preferences** ▶ **Publishing Engine**, returned by the ACL `username` function).

If this parameter is set to another value, then Arbortext Editor will treat that as the name of the HTTP query parameter. The value of it is then set to the value of the ACL `set pecompositionid` command option on the Arbortext Editor client. The `pecompositionid` option is available from Arbortext Editor **Tools** ▶ **Preferences** ▶ **Advanced**.

---

 **Note**

The `com.arbortext.e3.compositionIdentificationPolicy` parameter can be used for notifications on transactions that have not been queued, though its common use is to notify clients about queued transactions.

---

- `com.arbortext.e3.compositionEmailPolicy`

Specifies an HTTP query parameter name that will be used to provide an email address to Arbortext Publishing Engine. This is the parameter that will specify the email address where the Arbortext PE server will send an email notification, if a notifier is configured. The default value is explicitly set to `queue-email`.

Arbortext Editor clients will treat the value of this parameter as the name of the HTTP query parameter that specifies the user's email address. The value of the parameter is then set to the user's email address as displayed in **Notification Email Address** (**Tools** ▶ **Preferences** ▶ **Publishing Engine**).

The Arbortext Editor user specifies the email address using **Notification Email Address** or the ACL `set pecompositionemail` command option (available from Arbortext Editor **Tools** ▶ **Preferences** ▶ **Advanced**).

Refer to [Configuring a Notifier on page 97](#) for information on setting up an email notifier.

---

 **Note**

The `com.arbortext.e3.compositionEmailPolicy` parameter can be used for notifications on transactions that have not been queued, though its common use is to notify clients about queued transactions.

---

## The Global Debugging Parameters

The `debug` parameter logs information about transactions between clients and the server. However, you can set `debug` to include broader log levels. These parameters are applied after the `PE_HOME\e3\WEB-INF\classes\log4j.properties` file is processed.

- `debug`

Sets the log level for output messages. Explicitly set to `false` by default, which reports only fatal and error messages.

You can set it to `warn`, `info`, or `debug`.

Report data is sent to the servlet container or web application server log files. If you set the log level to include more data, be aware that the log files can become very large.

- `delete-temp`

Controls whether temporary storage for each Arbortext PE sub-process is deleted when it terminates. Explicitly set to `true` by default, which deletes the temporary storage upon termination. You can set it to `false` to preserve the Arbortext PE sub-process temporary storage for debugging. However, reset it to `true` when you are finished.

---

 **Caution**

Setting `delete-temp` to `false` will consume disk space quickly, so don't leave it set to `false` for long.

---

---

## The Global Application Logging Parameters

The application logging parameters control the type and level of logging for all applications running on Arbortext Publishing Engine. All the logging parameters follow the convention of the Java `log4j` package. For information on `e3log4j.properties` and how to use it, consult the standard documentation provided for `log4j`, available from:

[logging.apache.org/log4j/docs/manual.html](http://logging.apache.org/log4j/docs/manual.html)

The logging parameters that follow can take the following values:

- OFF
- FATAL
- ERROR
- WARN
- INFO
- DEBUG
- TRACE
- ALL
- INHERIT
- `com.arbortext.e3.applicationLog`  
Specifies the log level for all applications. By default, it's explicitly set to `WARN`, meaning messages are logged for `FATAL`, `ERROR`, and `WARN` levels.
- `com.arbortext.e3.applicationLog.compose`  
Specifies the log level for publishing requests submitted from a Arbortext Editor client. By default, it's explicitly set to `INHERIT`, which means use the setting for `com.arbortext.e3.applicationLog`.
- `com.arbortext.e3.applicationLog.convert`  
Sets the log level for requests submitted using the `f=convert` function. By default, it's explicitly set to `INHERIT`, which means use the setting for `com.arbortext.e3.applicationLog`.

After determining the logging level for an application, the Arbortext PE Request Manager and Arbortext PE sub-processes will ignore any attempt by the application to make log entries with a lower priority. For example, if the logging level is `WARN`, then messages with a level of `INFO`, `DEBUG`, and `TRACE` are ignored.

If the logging level is `INFO`, `DEBUG`, or `TRACE`, then intermediate files can be accepted and saved. Use the **Transaction Archive** link on the Arbortext Publishing Engine index page (described in [Monitoring and Reporting Using a Web Browser](#)



---

on page 21) to view the archived transactions. An archived transaction can include intermediate files if the application saves them. Refer to [Monitoring the Transaction Archive on page 115](#) for more information.

Learn how to use logging from custom applications in the *Programmer's Guide to Arbortext Publishing Engine*.

- `com.arbortext.e3.applicationLog.acl`  
Sets the log level for tracing in ACL applications
- `com.arbortext.e3.applicationLog.acl.package.function`  
Sets the log level for the ACL application specified by *package:function*
- `com.arbortext.e3.applicationLog.java.class`  
Sets the log level for the Java application specified by *class*
- `com.arbortext.e3.applicationLog.javascript.function`  
Sets the log level for the JavaScript application specified by *function*
- `com.arbortext.e3.applicationLog.vbscript.function`  
Sets the log level for the VBScript application specified by *function*

## Parameter Search Order

The Arbortext PE Request Manager starts with the most specific construct and proceeds to the broadest. For example, if a Java application is named `com.arbortext.petest.petest123`, Arbortext PE Request Manager determines the logging level by looking for the following configuration parameters in order:

- `com.arbortext.e3.applicationLog.java.com.arbortext.-petest.petest123`
- `com.arbortext.e3.applicationLog.java`
- `com.arbortext.e3.applicationLog`

Arbortext PE Request Manager applies the value of the first parameter specified in the `e3config.xml` file. After the log level is determined, attempts to make log entries with a lower priority are ignored.

## Application Log Output

An application log is always included in the set of files saved to a transaction archive. However, you can specify an additional location for sending application log messages:

- `com.arbortext.e3.applicationLog.display`  
By default, explicitly set to `false` to log messages to the transaction archive.

---

If set to `true`, application log messages are also written to the Arbortext Diagnostics window. See [Getting Trace Information on page 128](#) for more information.

- `com.arbortext.e3.applicationLog.displayPatternLayout`  
Specifies the Java `log4j` pattern to use when constructing log entries to standard output or the Arbortext Diagnostics window. The `com.arbortext.e3.applicationLog.display` must also be set to `true`.

The default is set to :

```
%d{HH:mm:ss} [%t] %-5p %c{2} - %m
```

Refer to the `log4j` documentation for an explanation of the pattern.

## The Global `com.arbortext.e3.epicInstallation` Parameter

The `com.arbortext.e3.epicInstallation` parameter specifies the path to the installation directory for Arbortext Publishing Engine. By default, Arbortext Publishing Engine assumes a relative path from the location of the `PE_HOME/e3/e3/WEB-INF/e3config.xml` file you're editing, which is `../..../...`. This parameter is not explicitly set in `e3config.xml`.

## The Global `com.arbortext.e3.tempFileDirectory` Parameter

The `com.arbortext.e3.tempFileDirectory` specifies where Arbortext Publishing Engine creates temporary files.

By default, Arbortext Publishing Engine assumes the location of `java.io.tmpdir`, which is a JVM system property. You can determine this location by clicking the **Java Properties** link on the Arbortext Publishing Engine index page. You can also view the location by clicking the `Status` link on the Arbortext Publishing Engine index page (refer to [Monitoring and Reporting Using a Web Browser on page 21](#) for information). This parameter is not explicitly set in `e3config.xml`. You shouldn't need to change the value.

The parameter values for `com.arbortext.e3.transactionDirectory` (described in [The Global Active Transaction Parameters on page 71](#)) and `com.arbortext.e3.transactionArchiveDirectory` (described in [The Global Transaction Archive Parameters on page 74](#)) are appended to this value.

---

## The Global `com.arbortext.e3.tempFilePrefix` Parameter

The `com.arbortext.e3.tempFilePrefix` parameter specifies a file name prefix for temporary files created by Arbortext PE Request Manager. A common prepended string makes the files easier to find because the files are grouped together in the temporary directory. By default, the value is explicitly set to `ati`, which prepends the string `ati` to the file names. You shouldn't need to change the value.

## The Global `com.arbortext.e3.tempFileSuffix` Parameter

The `com.arbortext.e3.tempFileSuffix` parameter specifies a file extension for temporary files created by Arbortext PE Request Manager. By default, the value is explicitly set to `.tmp`. You shouldn't need to change the value.

## Specifying a Request Handler

A **RequestHandler** processes each incoming HTTP request and has a `class` attribute specifying its associated Java class and an `id` attribute specifying its name. Each **RequestHandler** can have an associated list of parameters, corresponding to the implementation of the `com.arbortext.e3.E3RequestHandler` interface.

To support routing of HTTP and SOAP requests meeting specified criteria, you may need to define a corresponding `ClientFunction`. You would define one in the **AllowedFunctions** section to permit an application call to it. Refer to [Specifying the AllowedFunctions List on page 107](#) for more information.

## The RequestHandler for Arbortext Publishing Engine Functions

The Arbortext Publishing Engine Request Handler is implemented to support client functions that perform the work of processing a request. A client function refers to one either supplied by Arbortext or a custom application. Each request has a parameter that specifies a query specification resolving to a corresponding `f=function-name` or `f-function-name` function call. The parameters map each supported type of `function-name` to its underlying Java interface which handles the processing for Arbortext Publishing Engine.

The `RequestHandler` object specified in `e3config.xml` is `com.arbortext.e3.RequestHandler` with the ID `e3-functions`. This `RequestHandler` manages HTTP requests that meet one of the defined query conditions and are not otherwise routed by a previously invoked **RequestHandler**.

The `com.arbortext.e3.RequestHandler` has a list of parameters defining the query specification. In each case, an HTTP or SOAP request is evaluated for a match to assist in routing the request to the proper Arbortext PE sub-process pool. By default, parameters are associated with `ClientFunction` entries in the **AllowedFunctions** section of the `e3config.xml` file where permission for functions to run is specified.

In the following list of default parameters, the parameter name specifies the type of function call and the value specifies the associated Java interface that enables the underlying processing. Each of the Java classes in turn implements the **com.arbortext.e3.E3RequestFunction** interface. To implement a custom query parameter and support for the associated processing mechanism, you can review these parameter specifications for guidelines.

### Default Request Handler Parameters

name	value
query-function-name	f specifies the prefix for a function
function-prefix	f- also specifies a prefix for a function
f-acl and f-eval	<code>com.arbortext.e3.FunctionAcl</code> processes calls to ACL functions
f-convert	<code>com.arbortext.e3.FunctionNewConvert</code> processes calls to convert documents. It supports the built-in <code>convert</code> function call and its associated series of explicitly supported parameters (explained in the <i>Programmer's Guide to Arbortext Publishing Engine</i> ).
f-init	<code>com.arbortext.e3.FunctionInit</code> triggers a reload of custom ACL and JavaScript applications in an Arbortext PE sub-process pool. It supports the built-in <code>init</code> function call. Manual initialization is also available from a link on the Arbortext Publishing Engine index page (refer to <a href="#">Monitoring and Reporting Using a Web</a>

## Default Request Handler Parameters (continued)

name	value
	<a href="#">Browser on page 21</a> ).
f-java	com.arbortext.e3.Function Java processes calls to Java applications. It supports the built-in java application call and its associated class parameter specifying the Java class name.
f-javascript	com.arbortext.e3.Function Javascript processes calls to JavaScript functions. It supports the built-in javascript function call and its associated function parameter specifying the JavaScript function name.
f-vbscript	com.arbortext.e3.Function Vbscript processes calls to VBScript functions. It supports the built-in vbscript function call and its associated function parameter specifying the VBScript function name.
f-oldconvert	com.arbortext.e3.Function Convert processes calls to convert documents that use an older methodology. It supports the backward compatibility supplied by the oldconvert function call.
f-license	com.arbortext.e3.Function License requests a report about the Arbortext Publishing Engine license. It supports the built-in license function, which is an administrative tool that returns a license report. A license report is also available from a link on the Arbortext Publishing Engine index page (refer to <a href="#">Monitoring and Reporting Using a Web Browser on page 21</a> for information).
f-status	com.arbortext.e3.Function Status requests a report from the Arbortext PE Request Manager. It

## Default Request Handler Parameters (continued)

name	value
	supports the built-in <code>status</code> function, which is an administrative tool that returns an XHTML report. A status report is also available from a link on the Arbortext Publishing Engine index page.
<code>f-version</code>	<code>com.arbortext.e3.FunctionVersion</code> requests a report on your Arbortext Publishing Engine version. It supports the built-in <code>version</code> function, which is an administrative tool that returns a version report. A version report is also available from a link on the Arbortext Publishing Engine index page.
<code>f-comconfig-rescan</code>	<code>com.arbortext.e3.FunctionRescan</code> triggers a rescan of publishing configuration information used by Arbortext Editor clients. A rescan is also available from a link on the Arbortext Publishing Engine index page.
<code>f-supportdata</code>	<code>com.arbortext.e3.FunctionSupportData</code> collects data to put into a zip archive to submit to PTC Support. A zip archive is also available from a link on the Arbortext Publishing Engine index page.
<code>q-enable</code>	<code>com.arbortext.e3.function.QueueEnable</code> enables queuing. The queuing functions <code>f=q-queue-function</code> can be used by a custom application to send requests to a Queue Manager. These functions are described in the <i>Programmer's Guide to Arbortext Publishing Engine</i> .
<code>q-holdall</code>	<code>com.arbortext.e3.function.QueueHold</code> holds all queued transactions. The queuing functions <code>f=</code>

## Default Request Handler Parameters (continued)

name	value
	<code>q-queue-function</code> can be used by a custom application to send requests to a Queue Manager.
<code>q-list</code>	<code>com.arbortext.e3.function.QueueList</code> enables listing queues. The queuing functions <code>f=q-queue-function</code> can be used by a custom application to send requests to a Queue Manager.
<code>qt-cancel</code>	<code>com.arbortext.e3.function.TransactionCancel</code> cancels a queued transaction. The queuing functions <code>f=qt-queued-transaction-function</code> can be used by a custom application to send requests to a Queue Manager. These functions are described in the <i>Programmer's Guide to Arbortext Publishing Engine</i> .
<code>qt-discard</code>	<code>com.arbortext.e3.function.TransactionDiscard</code> discards a queued transaction. The queuing functions <code>f=qt-queued-transaction-function</code> can be used by a custom application to send requests to a Queue Manager.
<code>qt-execute</code>	<code>com.arbortext.e3.function.TransactionExecute</code> executes a queued transaction. The queuing functions <code>f=qt-queued-transaction-function</code> can be used by a custom application to send requests to a Queue Manager. These functions are described in the <i>Programmer's Guide to Arbortext Publishing Engine</i> .
<code>qt-hold</code>	<code>com.arbortext.e3.function.TransactionHold</code> holds a queued transaction. The queuing functions <code>f=</code>

## Default Request Handler Parameters (continued)

name	value
	<code>qt-queued-transaction-function</code> can be used by a custom application to send requests to a Queue Manager.
<code>qt-list</code>	<code>com.arbortext.e3.function.TransactionList</code> lists all queued transactions. The queuing functions <code>f=qt-queued-transaction-function</code> can be used by a custom application to send requests to a Queue Manager. These functions are described in the <i>Programmer's Guide to Arbortext Publishing Engine</i> .
<code>qt-move</code>	<code>com.arbortext.e3.function.TransactionMove</code> moves a queued transaction. The queuing functions <code>f=qt-queued-transaction-function</code> can be used by a custom application to send requests to a Queue Manager. .
<code>qt-retrieve</code>	<code>com.arbortext.e3.function.TransactionRetrieve</code> retrieves a queued transaction. The queuing functions <code>f=qt-queued-transaction-function</code> can be used by a custom application to send requests to a Queue Manager.



## Default Request Handler Parameters (continued)

name	value
qt-setpriority	om.arbortext.e3.function.TransactionPriority sets the priority of a queued transaction. The queuing functions <code>f=qt-queued-transaction-function</code> can be used by a custom application to send requests to a Queue Manager.
qt-status	com.arbortext.e3.function.TransactionStatus gets the status of a queued transaction. The queuing functions <code>f=qt-queued-transaction-function</code> can be used by a custom application to send requests to a Queue Manager.

For information about writing and implementing custom Java, JavaScript, VBScript, and ACL applications, consult the *Programmer's Guide to Arbortext Publishing Engine*.

## Specifying Request Selectors

A **RequestSelector** has a `class` attribute specifying its associated Java class and an `id` attribute specifying its identifying name. Its purpose is to test an incoming HTTP request and help determine its routing. Each `RequestSelector` can have an associated list of parameters, as defined and used by the application that implements the `com.arbortext.e3.E3RequestSelector` interface.

An Arbortext PE sub-process pool can use one or more defined `RequestSelector` tests to determine whether it should handle the request. You can create sophisticated tests for routing a request to a specific Arbortext PE sub-process pool by combining them using `And` or `Or` logic in a **Test Set** configured in a sub-process pool.

There are several `RequestSelector` classes defined:

- class `com.arbortext.e3.TestHeaderMatch`, ID `test-text-xml`  
Determines if an HTTP request has a particular message header and that the header value matches a specified pattern.

The parameters `header-name` and `header-pattern`, defined as

---

Content-type and text/xml, detect a Content-type: text/xml header.

- class com.arbortext.e3.TestQueryMatch, ID test-f-java  
Determines if an HTTP request has a query name and query pattern that match a specified name and pattern.

The parameters query-name and query-pattern, defined as f and java, detect f=java or f-java requests.

- class com.arbortext.e3.TestQueryMatch, ID test-convert  
Determines if an HTTP request has a query name and query pattern that match a specified name and pattern.

The parameters query-name and query-pattern, defined as f and convert, detect f=convert document conversion requests.

- class com.arbortext.e3.TestQueryMatch, ID test-tie  
Determines if an HTTP request has a query name and query pattern that match a specified name and pattern.

The parameters query-name and query-pattern, defined as class and com.arbortext.e3c.\*, detect Arbortext Publishing Engine publishing requests that originate from Arbortext Editor clients.

- class com.arbortext.e3.TestQueryMatch, ID test-wvs-class  
Determines if an HTTP request has a query name and query pattern that match a specified name and pattern.

The parameters query-name and query-pattern, defined as class and com.arbortext.ptc.windchill.Compose, detect Arbortext Publishing Engine publishing requests that originate from Windchill Visualization Service (WVS). The WVS uses Arbortext Publishing Engine to publish XML and SGML documents stored in Windchill to outputs such as PDF and HTML.

For further information on WVS configuration and usage with Arbortext Publishing Engine, refer to *Configuring Arbortext Publishing Engine for the Windchill Visualization Service*. For more information about the WVS, refer to the *Windchill Business Administrator's Guide*.

- class com.arbortext.e3.TestQueryMatch, ID test-archive-transaction  
Determines if an HTTP request has a query name and query pattern that match a specified name and pattern.

---

The parameters `query-name` and `query-pattern`, defined as `archive-transaction` and `yes`, detect `archive-transaction` requests.

- class `com.arbortext.e3.TestQueryMatch`, ID `test-alternate-transaction-archive`

Determines if an HTTP request has a query name and query pattern that match a specified name and pattern.

The parameters `query-name` and `query-pattern`, defined as `alternate-transaction-archive` and `yes`, detect `alternate-transaction-archive` requests.

## Specifying Test Sets

A `TestSet` can specify one or more `Test` names used to filter the HTTP request. A `Test` refers to a `RequestSelector` ID of the same name that specifies what to filter in the HTTP request. A Queue Manager, a queue, an Arbortext PE sub-process pool and a defined notifier can each specify a `TestSet`. You can create sophisticated tests for routing a request by combining `Tests` using optional `And` or `Or` logic within a `TestSet`.

For example, you could implement a sub-process pool dedicated to fulfilling Arbortext Editor publishing requests. A sub-process pool with `id="pool-tie"` could define two `Tests` in a `TestSet`, called `test-f-java` and `test-tie`. Use `And` logic, so that both must be present to succeed. The `Test` names refer to configured **RequestSelector** of the same name, `test-f-java` and `test-tie`, defined as follows:

```
<RequestSelector class="com.arbortext.e3.TestQueryMatch" id="test-f-java"
  <Parameter name="query-name" value="f"/>
  <Parameter name="query-pattern" value="java" />
</RequestSelector>
RequestSelector class="com.arbortext.e3.TestQueryMatch" id="test-tie">
  <Parameter name="query-name" value="class"/>
  <Parameter name="query-pattern" value="com.arbortext.e3c.*" />
</RequestSelector>
```

Then the sub-process pool would define the `TestSet` to specify these tests:

```
<TestSet>
  <And>
    <Test name="test-f-java" />
    <Test name="test-tie" />
  </And>
</TestSet>
```

---

A couple of test sets are configured in `e3config.xml` by default for archiving transactions. If a transaction matches the criteria in one of the test sets, then the transaction will be archived (provided `com.arbortext.e3.transactionArchive.enable` is set to `true`):

- The test for archiving transactions that are made available through the **Transaction Archive** link on the Arbortext Publishing Engine index page:

```
<TestSet id="archive-transaction-test">
  <Test name="test-archive-transaction"/>
</TestSet>
```

The transaction archive parameter that identifies the ID of the test set for the query to match:

```
<Parameter name="com.arbortext.e3.transactionArchive.testSet"
  value="archive-transaction-test" />
```

The query parameter looks for a match on the request for `archive-transaction=yes`.

- The test for archiving transactions that are stored in an alternate location (not made available through the **Transaction Archive** link):

```
<TestSet id="archive-transaction-to-alternate-location-test">
  <Test name="test-alternate-transaction-archive" />
```

The transaction archive parameter that identifies the ID of the test set for the query to match:

```
<Parameter name="com.arbortext.e3.transactionArchive.
  alternateLocation.testSet"
  value="archive-transaction-to-alternate-location-test" />
```

The query parameter looks for a match on the request for `alternate-transaction-archive=yes`.

For further information about configuring a **RequestSelector**, refer to [Specifying Request Selectors on page 89](#).

## Specifying Caches

A Cache Manager stores something temporarily in anticipation of a request to use it, which can improve processing performance. The Cache Manager defines what to cache and when to cache it. Each **CacheManager** must implement the **com.arbortext.e3.E3CacheManager** interface.

A **CacheManager** object has been implemented to manage publishing configuration conversion. The class `com.arbortext.e3.CompConfigCache`, ID `compconfig-cache` caches configuration information for Arbortext Editor clients that use Arbortext Publishing Engine as a publishing server.

---

## Specifying Queues

The `queue` element has several configuration parameters that control the following:

- A queue can be enabled or disabled. If it is enabled, it can be active or inactive.

When a queue is enabled, it can accept transaction requests. However, it can only allow the Queued Transaction Scheduler to execute transactions when it is active.

When a queue is disabled, it can still accept transaction requests. However, no transactions can be executed by the Queued Transaction Scheduler, even if the queue is active. A disabled queue must be enabled to make its requests available for execution; however, those transactions can only be processed during an active period.

The queue's startup state can be configured (see [The initial-state Attribute on page 95](#)). However, an administrator can explicitly enable or disable a queue from the Queue List web page (see [Requesting the Queue Reports on page 118](#)).

- A queue can be configured to be active only during specified periods (see [The active-interval Attribute on page 94](#)). An active queue supplies transactions to the Queued Transaction Scheduler. An inactive queue only accepts transactions, but it does not allow Queued Transaction Scheduler to execute them until it becomes active. An active queue must also be enabled to process its transactions.
- A queue can be required to wait until other queues have either started or finished their transactions (see [The previous-queues Attribute on page 96](#)).
- A transaction can have a priority assigned to it within a queue (see the `queue-priority` parameter in [Queuing Query Parameters on page 46](#)).

Transaction priorities are 1 through 5, highest to lowest, which can be designated by the incoming request. Transactions assigned the same priority within a queue are processed first in, first out. If no priority is assigned, it defaults to 3.

- A transaction can be executed when the previous transaction is completed, after all previous transactions have started, or must execute even if previous transactions are not able to run. Execution criteria is applied to each priority group (see [The scheduling-option Attribute on page 96](#)).
- A hold can be placed on individual transactions within a queue or on every transaction in the queue. A hold will persist each time Arbortext Publishing

---

Engine starts. Refer to [The Transaction List Page on page 119](#) and [The hold-all Attribute on page 95](#) for information.

- A transaction can be executed ahead of all other transactions in the system, overriding any other configuration settings that might prevent it.
- The number of queued transactions allowed to execute simultaneously overall is controlled by a global parameter `maxConcurrentQueuedTransactions` (described in [The Global Active Transaction Parameters on page 71](#)) and by the `max-concurrent-transactions` queue attribute for individual queues (described in [The max-concurrent-transactions Attribute on page 96](#)).
- The number of Arbortext PE sub-processes allocated simultaneously for processing queued transactions within a pool is controlled by the `maxConcurrentQueuedTransactions` parameter, described in [The maxConcurrentQueuedTransactions Attribute on page 102](#).
- An Arbortext PE sub-process pool can be configured to be a dedicated queuing pool, meaning it only accepts queued requests. See [Specifying a Dedicated Queuing Pool on page 106](#).

Refer to [The e3config.xml Configuration File on page 71](#) for more information on how to change attributes.

## The active-interval Attribute

Specifies the times and days during which a queue is active and its transactions can be executed. The default is the empty string "", which means the queue is always active.

Enter a list of time periods separated by semicolons, using the form:

`p1;p2;p3`

A period can be one of the following:

`sunday, monday, tuesday, wednesday, thursday, friday, saturday, sunday, weekday, or weekend`

You can also include the time of day by adding a comma after the day and then specifying the time range, using the form:

`HH:MM-HH:MM`

A time range consists of a 24 hour specification. If the second `HH:MM` specification is earlier than the first, then it specifies a time on the following day. If the second `HH:MM` is omitted, it defaults to `23:59`.

For example:

- `Parameter name="active-interval" value="weekday"`

---

The queue is active all day Monday through Friday but not on weekends.

- Parameter name="active-interval" value="weekday,20:00-06:00;weekend"


The queue is active Monday through Friday from 8 p.m. to 6 a.m. the next morning and all day on weekends.

- Parameter name="active-interval" value="monday,12:00-16:00;wednesday,17:00"

The queue is active on Mondays from 12 p.m. to 4 p.m. and on Wednesdays from 5 PM to midnight. The end of the day is implied when no end time is specified.

## The hold-all Attribute

Specifies whether transactions placed on the queue should be automatically held.

If set to `yes`, all transactions on the queue will be marked `Hold` using  on the Transaction List page for **Queued Transaction List**. The default `no` means that transactions are not held automatically and are available to be processed. A hold persists each time Arbortext Publishing Engine starts. A queue may be enabled and active even if all its transactions are being held, but no transactions will be processed. Individual transactions can be held or released by the administrator using the Hold actions described in [The Transaction List Page on page 119](#).

## The initial-state Attribute

Specifies the state of a queue.

- `enabled`  
specifies that the queue should always be enabled.
- `disabled`  
specifies that the queue should always be disabled.
- `saved` (the default)  
specifies that the state should be set to what it was the last time the Arbortext PE server was shut down.

The state of the queue can be changed by the administrator using the Queue List link on the Arbortext Publishing Engine index page. See [Requesting the Queue Reports on page 118](#) for information.

---

## The max-concurrent-transactions Attribute

Specifies the maximum number of transactions from the queue that the Queued Transaction Scheduler may run simultaneously. Specify a number or zero. The default value, 0, means there is no limit.

## The previous-queues Attribute

Specifies a list of queue IDs separated by commas. The default is the empty string "", which means there are no dependencies on other queues. If you specify one or more queue IDs, transactions on this queue will not execute until each of those queues are inactive, disabled, or empty except for transactions that are being held. Separate a list of queues using commas:

```
Parameter name="previous-queues" value="queue3,queue5"
```

## The scheduling-option Attribute

Specifies how queued transactions should be executed.

- `strict-complete`  
No transaction shall start executing until all previous transactions on the queue have finished executing, including held transactions. Note that choosing this value means that transactions from this queue will never execute in parallel.
- `strict-parallel`  
No transaction shall start executing until all previous transactions on the queue have started executing.
- `relaxed` (the default)  
A transaction may start executing if earlier transactions on the queue are not able to execute for some reason.

By default, the `relaxed` approach is to allow the Queued Transaction Scheduler to try executing transactions in order, but place a higher emphasis on keeping busy and executing transactions rather than ensuring that transactions are processed in order.

### Example

A queue contains three transactions, in order: A, B, and C. The Queued Transaction Scheduler scans the queue looking for a transaction to execute. Transaction A can't be executed for some reason (it might be held or it might be waiting for an Arbortext PE sub-process to become available from a specific pool). The Queued Transaction Scheduler would obey the queue's configuration for strictness:



- 
- If a queue is configured `strict-complete` or `strict-parallel`, the Queued Transaction Scheduler would not execute transaction B, because A could not run.
  - If the queue is configured `relaxed`, then the scheduler would execute transaction B instead.

### Example

A queue contains three transactions, in order: D, E, and F. The Queued Transaction Scheduler scans the queue searching for a transaction to execute. Transaction D is executing. If the queue is configured `strict-complete`, transaction E can't run because D is not finished. If the queue is configured `strict-parallel` or `relaxed`, transaction E could be started.

## Queuing Function Reference

The queuing functions (`f=q-queue-function` and `f=qt-queued-transaction-function`) can be used by a custom application to send requests to a Queue Manager. These functions are described in the *Programmer's Guide to Arbortext Publishing Engine*.

## Configuring a Notifier

You can configure a transaction notifier to report the progress of a queued transaction. An example is a notifier that can send email notification to a client when a queued transaction has been completed.

Notifiers are configured in `e3config.xml`. A notifier accepts parameters that control its operation, as well as an optional `TestSet` to respond only to certain types of transactions. A notifier Java object implements the class `com.arbortext.e3.E3Notifier`, found in:

`PE_HOME\lib\classes\pecommon.jar`

A sample Java notifier is available in:

`PE_HOME\e3\samples\java\com\arbortext\e3\queue\MailNotifier.java`

For more information on using JavaMail, consult the JavaMail documentation at:

<http://java.sun.com/products/javamail>

A notifier can send an email message when a transaction meets the following criteria:

- The request contains a query parameter that matches the notifier's `TestSet` specification, if one is specified.
- The request includes a query parameter that matches the value of the notifier's `request-parameter`

---

The message will be sent to the email address provided using the parameter name specified by `request-parameter` and the email address specified by the parameter's value.

The following parameters can be specified in `e3config.xml` to control the behavior of the sample notifier. All parameters except `request-parameter` and `target-states` will be passed to the JavaMail package.

- `com.arbortext.e3.request-parameter` (required)

Specifies the name of a request parameter. The notifier will only send email about state changes in transactions whose requests include this parameter's value. Email will be sent to the address given by the value of this parameter. There is no default value.

```
Parameter name="request-parameter" value="queue-email"
```

The HTTP query must then include a parameter such as:

```
queue-mail="user@host"
```

In the query, you would replace the value of `user` and `host` with the email address where you want to send notification.

- `com.arbortext.e3.target-states` (required)

Specifies which states are of interest in a comma-separated list of state names. Case and white space are ignored. There is no default value.

- `initializing`
- `waiting`
- `queued`
- `processing`
- `complete`
- `cancelled`

States are explained in [Transaction States on page 39](#).

- `mail.from` (required)

Specifies the return address for each email message. Used by the `InternetAddress.getLocalAddress` method to specify the email address.

- `mail.host` (optional)

Specifies the default Mail server, usually in the form *mailhost.yourcompany.com*.

The default value is the `localhost`, so you will likely need to specify your mail host.

- `mail.transport.protocol` (optional)

---

Specifies the protocol to use for sending email, for example, IMAP. The default is SMTP.

You can use `mail.debug` to specify the debug mode for troubleshooting. Setting this to `true` will turn on debug mode. The default value is `false`, which turns it off. You can also use the `JavaMailSession.setDebug` method to control the debug mode.

Depending on the protocol being used, the system could be prompted for more information or you may have more than one type of supported protocol. However, it's not likely you will need to implement any of the following on your production system. Be sure you completely understand what is required for your protocol before using these parameters.

- `mail.user` (optional)

Specifies the user name for connecting to the email server. The default value is `mail.user`.

The `JavaMail Transport` object has a `connect` method that uses this property to obtain the user name, if the `mail.protocol.user` property is not supplied.

- `mail.protocol.host` (optional)

Specifies the Mail server using the specified protocol. If not specified, the value falls back to the `mail.host` property and SMTP is presumed. For example, you could specify IMAP for `protocol`.

- `mail.protocol.user` (optional)

Specifies the protocol to use for the specified user when connecting to the Mail server. If not specified, the value falls back to `mail.user` property.

## Sample Notifier

In the `e3config.xml` file, a sample notifier for email has been configured in a commented section. Look for:

```
<Notifier class="com.arbortext.e3.queue.MailNotifier" id="notifier1">
<Parameter name="mail.host" value="mailhost.yourcompany.com" />
<Parameter name="mail.from" value="test@yourcompany.com" />
<Parameter name="target-states" value="complete,processing" />
<Parameter name="request-parameter" value="queue-email" />
</Notifier>
```

You would remove the comment markers and replace the values with the mail server information for your site. For example:

```
<Notifier class="com.arbortext.e3.queue.MailNotifier" id="notifier1">
<Parameter name="mail.host" value="int-mail.acme.com" />
<Parameter name="mail.from" value="job-update@acme.com" />
<Parameter name="target-states" value="complete,queued" />
<Parameter name="request-parameter" value="queue-email" />
</Notifier>
```

---

Then, be sure to set the `compositionEmailPolicy` parameter (see [The Global Queuing Parameters on page 76](#) for information.):

```
Parameter name="com.arbortext.e3.compositionEmailPolicy"  
  value="queue-email" />
```

The notifier would respond to a query that included a parameter like the following:

```
name="queue-mail" value="user@host"
```

Where *user@host* is the email where the notification should be sent. This query can originate from a

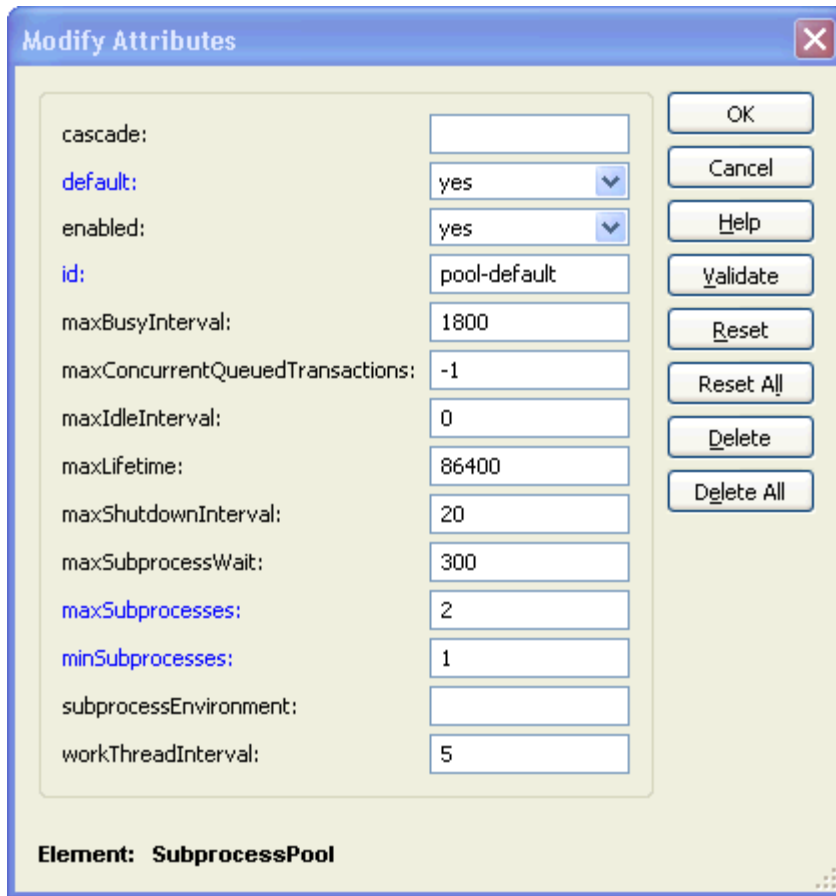
Arbortext Editor clients would supply this value through their **Notification Email Address** in **Tools ▶ Preferences ▶ Publishing Engine**. The **Notification Email Address** is included in the publishing request.

## Configuring Sub-process Pools

A **SubprocessPool** has an `id` attribute specifying its identifying name. Each **SubprocessPool** can have an associated list of parameters. Each **SubprocessPool** also specifies a **Test Set** that evaluates HTTP requests and accepts those that meet the test criteria. A request is offered to each Sub-process pool defined in `e3config.xml` in the order in which it appears.

Custom **SubprocessPool** objects are not supported in this release.

The default **SubprocessPool** must be defined. It should always be defined last in the list as the **SubprocessPool** objects are evaluated in the order specified in the `e3config.xml` file. Each predefined **SubprocessPool**, including the default, is explained in more detail in the following sections. The dialog box where you set the **SubprocessPool** attributes looks like the following:



Refer to [The e3config.xml Configuration File on page 71](#) for more information on how to change attributes.

The **SubprocessPool** objects explicitly specified in `e3config.xml` are:

- ID `pool-wvs`  
This sub-process pool processes all requests from the Windchill Visualization Service (WVS). This pool is disabled by default, so you need to enable this pool to use it.
- ID `pool-default`  
This sub-process pool is specified as the `defaultpool`, and it processes all requests not processed by previously defined pools. It also can accept their overflow if their `cascade` attribute is set. This pool is enabled by default. It does not specify a `TestSet` so that it can accept any request not directed to another pool. It must be the last pool defined in the `e3config.xml` file.

---

## The cascade Attribute

Specifying `cascade` provides the name of the Arbortext PE sub-process pool to which a request can be routed if all the Arbortext PE sub-processes in this pool are busy.

## The default Attribute

Specifying `default` signifies whether this Arbortext PE sub-process pool is the default pool.

## The enabled Attribute

Specifying `enabled` determines whether this Sub-process pool is enabled to take requests.

## The id Attribute

Specifying `id` provides the unique identifier of the Arbortext PE sub-process pool.

## The maxBusyInterval Attribute

Specifying `maxBusyInterval` sets the maximum time in seconds that an Arbortext PE sub-process is assumed to be busy processing a request. This attribute allows Arbortext Publishing Engine to terminate an Arbortext PE sub-process that is no longer accessible due to an error condition. The default value is 1800 seconds, or thirty minutes. Setting the value to 0 means there is no limit.

---

### Caution

It's possible that an Arbortext PE sub-process could be terminated while it's processing a lengthy job. When you set `maxBusyInterval`, estimate a value that's higher than the time it would take to process the most extensive request you might submit.

---

## The maxConcurrentQueuedTransactions Attribute

Specifying `maxConcurrentQueuedTransactions` sets the maximum number of queued transactions that may be processed simultaneously by the Arbortext PE sub-process pool.

There are two other possible values, `-1` and `0`.

---

If it's set to `-1`, the default, there is no specified limit, and the Arbortext PE sub-process pool will execute as many queued transactions as it can by keeping all Arbortext PE sub-processes busy.

If it's set to `0`, no queued transactions will be accepted by the Arbortext PE sub-process pool, effectively reserving the entire pool for immediate transactions only. If you need to reserve a Sub-process pool for queued requests, refer to [Specifying a Dedicated Queuing Pool on page 106](#).

## The `maxIdleInterval` Attribute

Specifying `maxIdleInterval` sets the maximum time in seconds Arbortext Publishing Engine allows an Arbortext PE sub-process to be idle. If an Arbortext PE sub-process is idle for the specified time, it's terminated.

The `maxIdleInterval` value only applies if `maxSubprocesses` is greater than `minSubprocesses` and there are more Arbortext PE sub-processes running than the number specified by `minSubprocesses`.

## The `maxLifetime` Attribute

Specifying `maxLifetime` sets the maximum time in seconds that an Arbortext PE sub-process can remain in the pool before being terminated to recycle the service.

## The `maxShutdownInterval` Attribute

Specifying `maxShutdownInterval` sets the maximum number of seconds that Arbortext Publishing Engine will wait to allow an Arbortext PE sub-process to shut down on its own. If an Arbortext PE sub-process has been instructed to shut down by the Arbortext PE Request Manager and it's still running after this period of time, it will be terminated.

## The `maxSubprocesses` Attribute

Specifying `maxSubprocesses` sets the maximum number of Arbortext PE sub-processes in the service pool. You can minimize the overhead associated with launching and terminating Arbortext PE sub-processes by setting `maxSubprocesses` and `minSubprocesses` to the same number. There is no physical limit to the maximum number you can set, but be aware of the overhead required to run more than you really need to handle the request load.

In most cases, you will achieve the most efficient throughput by setting `maxSubprocesses` to the number of processors on the Arbortext Publishing Engine server system. This guideline ensures all CPUs will be used and avoids unnecessary context switching, which can hinder throughput.

---

## The `maxSubprocessWait` Attribute

Specifying `maxSubprocessWait` sets the maximum time in seconds that Arbortext Publishing Engine will wait for an Arbortext PE sub-process to become available to process a request. If the time limit is exceeded, it returns a message that all Arbortext PE sub-processes are busy.

## The `minSubprocesses` Attribute

Specifying `minSubprocesses` sets the number of Arbortext PE sub-processes automatically started when the Arbortext PE Request Manager gets its first request. You can minimize the overhead associated with launching and terminating Arbortext PE sub-processes by setting `maxSubprocesses` and `minSubprocesses` to the same number.

## The `subprocessEnvironment` Attribute

You can specify `subprocessEnvironment` to have Arbortext Publishing Engine set the environment variables of the Arbortext PE sub-processes. To ensure that Arbortext PE sub-processes have the expected values for environment variables, set them using the `subprocessEnvironment` parameter for each Arbortext PE sub-process pool in the `e3config.xml`.

If you are using the Tomcat servlet container, be aware that newer versions clear some environment variables, such as `CLASSPATH`, and then set them to values needed by Tomcat prior to starting the Tomcat servlet container.

Any other variables in the Arbortext Publishing Engine environment are passed to its Arbortext PE sub-processes unchanged.

## The `workThreadInterval` Attribute

Specifying `workThreadInterval` sets an interval in *integer* seconds for Arbortext Publishing Engine to check the status of the Arbortext PE sub-processes. Arbortext Publishing Engine can check for the following:

- An Arbortext PE sub-process has started and is ready to process a request
- An Arbortext PE sub-process has been idle for too long and should be terminated
- An Arbortext PE sub-process has been running for too long and should be terminated
- An Arbortext PE sub-process is presumed to be hung, should be terminated, and another one started if needed

As each check is made, Arbortext Publishing Engine takes appropriate action according to its attributes. The default value is 5 seconds.



---

## SubprocessContext Parameter

There is one parameter for the SubprocessContext of the SubprocessPool, `com.arbortext.e3.initialScript`:  
Parameter name="com.arbortext.e3.initialScript" value="filepath"

`initialScript` allows you to set environment variables in an ACL script before loading information from the custom directory. For example, you could have different custom directories for each Sub-process pool because `initialScript` is processed before the custom directory.

For example:

```
<SubprocessPool id="pool-special-script" enabled="yes" >
  <SubprocessContext>
    <Parameter name="com.arbortext.e3.initialScript"
      value="d:\custom\script-init.acl" />
  </SubprocessContext>
  <TestSet>
    <Test name="script-init" />
  </TestSet>
</SubprocessPool>
```

You would use the ACL file specified by `initialScript` to set something like:

```
main::ENV['APTCUSTOM']='D:\special_custom'
```

## The Windchill Visualization Service Sub-process Pool

The Windchill Visualization Service (WVS) sub-process pool (ID `pool-wvs`) services all requests to Arbortext Publishing Engine from the WVS. The WVS uses Arbortext Publishing Engine to publish XML and SGML documents stored in Windchill PDMLink to outputs such as PDF and HTML. The WVS **Test Set** detects requests that can be processed by this sub-process pool.

The Windchill Visualization Service sub-process pool defines a **Test** in the **Test Set** called `test-wvs-class`, which refers to a WVS **RequestSelector** of the same name.

The Windchill Visualization Service sub-process pool is disabled by default. For further information on WVS configuration and usage with Arbortext Publishing Engine, refer to the Arbortext Editor help topic *Configuring Arbortext Publishing Engine for the Windchill Visualization Service*. For more information about the WVS, refer to the *Windchill Business Administrator's Guide*.

---

## The Default Sub-process Pool

The default Sub-process pool (ID `pool-default`) is a dedicated pool to process any HTTP requests not caught by the other preceding ones. There is no **Test Set** to filter requests, so the default pool must attempt to service a request or return an error explaining a request can't be serviced. One `pool-default` must always be defined to handle requests, though it can work alone or as a cascade pool if it's defined last in `e3config.xml`.

## Specifying a Dedicated Queuing Pool

If you need to reserve a Sub-process pool for queued requests, you can define a test that will accept only queued requests by using the same Test Set as the Queue Manager. In the following example, the subprocess pool will not accept immediate requests, and its sub-processes will be dedicated to process queued transactions.

```
<RequestSelector class="com.arbortext.e3.TestQueryMatch"
  id="test-queue">
  <Parameter name="query-name" value="queue"/>
  <Parameter name="query-pattern" value="yes"/>
</RequestSelector>
```

Then define a **Test** in a Sub-process pool **Test Set**, called `test-queue`:

```
<SubprocessPool id="pool-queue" enabled="yes" >
  <TestSet>
  <Test name="test-queue" />
  </TestSet>
</SubprocessPool>
```

If you need to reserve a Sub-process pool for immediate requests, see [The `maxConcurrentQueuedTransactions` Attribute on page 102](#).

## Specifying a Sub-process Pool for Arbortext Editor Clients

Arbortext Editor users can run as clients of Arbortext Publishing Engine and ask Arbortext Publishing Engine to perform their document publishing for all outputs. You might want to dedicate a sub-process pool to fulfill Arbortext Editor publishing requests.

Set the sub-process pool ID `id="pool-tie"`. Then define two **Tests** in the **Test Set**, called `test-f-java` and `test-tie`. The **Test** names refer to **RequestSelector** objects already defined, named `test-f-java` and `test-tie`. They're specified using AND logic, so both must be met to succeed. The `test-f-java` and `test-tie` tests will detect a query match for these specifications in the HTTP request .

```
<SubprocessPool id="pool-tie" cascade="pool-default" enabled="yes"
  maxSubprocesses="2" minSubprocesses="1"
  <TestSet>
  <And>
```

```
<Test name="test-f-java" />
<Test name="test-tie" />
</And>
</TestSet>
</SubprocessPool>
```

This sub-process pool would process all requests from Arbortext Editor clients requesting publishing operations. You need to set `enabled` to `yes` to make this pool available, otherwise, Arbortext Editor client requests will go to the default pool. If the Arbortext PE sub-processes in this pool are busy, the overflow can cascade into the default pool by setting `cascade` to `pool-default`. For information about configuring a **RequestSelector**, refer to [Specifying Request Selectors on page 89](#).

## Specifying the AllowedFunctions List

Custom applications must be loaded in an Arbortext PE sub-process pool. Writing custom applications is covered in the *Programmer's Guide to Arbortext Publishing Engine*. However, to make custom applications available for client requests, you must specify access to them in the **AllowedFunctions** list.

The `ClientFunction` for Java allows any classes matching the wildcard specification **com.arbortext.\*** to be executed. If you have Java interfaces that use another naming convention, you can add another `ClientFunction` in the `AllowedFunctions` section to permit the application calls.

A `ClientFunction` has a `pattern` attribute specifying a query function or application, as well as a `type` attribute specifying its underlying programming language. Each `ClientFunction` can have an associated list of attributes. The attributes are those used by the application which implements the **com.arbortext.e3.E3RequestFunction** interface.

There are several `ClientFunction` objects already specified in the file:

- `pattern e3apptest:testapp` with a `type` of `acl`  
`pattern e3apptest2:testapp` with a `type` of `acl`

These specifications enable sample ACL function test links on the Arbortext Publishing Engine index page runs it (refer to [Monitoring and Reporting Using a Web Browser on page 21](#) for information). In your production environment, you may want to remove the sample function from the list.

- `pattern e3appsave:appsave` with a `type` of `acl`.

If you are troubleshooting a particular document or implementation, you should load it in Arbortext Publishing Engine Interactive and then choose **Tools ► Save Application**.

- `pattern for Service Information Manager main::composeSisPE` with a `type` of `acl`

---

Enables publishing from Windchill Service Information Manager.

- pattern for any Java class matching `com.arbortext.*` with a type of `java`

pattern for any Java class matching `com.ptc.arbortext.*` with a type of `java`

These specifications allow all Java classes delivered with Arbortext Publishing Engine, including the Java application test link on the Arbortext Publishing Engine index page (refer to [Monitoring and Reporting Using a Web Browser on page 21](#) for information).

- pattern `E3AppTest` with a type of `javascript`

By default, the sample JavaScript function test link on the Arbortext Publishing Engine index page runs it (refer to [Monitoring and Reporting Using a Web Browser on page 21](#) for information). In your production environment, you may want to remove the sample function from the list.

- pattern specifying the `E3AppTest` with a type of `vbscript`

By default, the sample VBScript function test link on the Arbortext Publishing Engine index page runs it (refer to [Monitoring and Reporting Using a Web Browser on page 21](#) for information). In your production environment, you may want to remove the sample function from the list.

## How to Deploy Custom Applications

ACL function files are placed in the `PE_HOME/custom/init` subdirectory, or placed in the `PE_HOME/custom/scripts` directory and loaded from a script in `PE_HOME/custom/init`. The HTTP request sent to Arbortext Publishing Engine would specify the `f=acl` query and its `function=function-name` parameter that you define in this section.

Java `.class` or `.jar` files are placed in the `PE_HOME/custom/classes` subdirectory. The HTTP request sent to Arbortext Publishing Engine would specify the `f=java` function and its `class=class-name` parameter that you define in this section.

JavaScript function files are placed in the `PE_HOME/custom/init` subdirectory, or placed in the `PE_HOME/custom/scripts` directory and loaded from a script in `PE_HOME/custom/init`. The HTTP request sent to Arbortext Publishing Engine would specify the `f=javascript` function and its `function=function-name` parameter that you define in this section.

---

 **Note**

Arbortext Publishing Engine supports only the Rhino type of JavaScript files. You need to specify the type of JavaScript interpreter in the JavaScript file or use the `set javascriptinterpreter ACL` command in an initialization file placed in `custom/init` to specify the Rhino type of JavaScript.

---

## Specifying Initializers

Initialization actions can be automatically launched if they're defined in `e3config.xml`. Each **Initializer** must implement the `com.arbortext.e3.E3Initializer` interface.

Setting the `defer` value to `yes` means that Arbortext Publishing Engine can take requests even if the initializer hasn't finished loading yet.

Two **Initializer** objects are implemented to specifically support Arbortext products that interact with Arbortext Publishing Engine:

- `class com.arbortext.e3.CompConfigInit` with ID `CompConfigInit`. This initializer obtains and caches a Publishing Configuration report for Arbortext Editor clients using Arbortext Publishing Engine as a publishing server.
- `class com.arbortext.e3.queue.QueueSchedulerInit` with ID `QueueSchedulerInit`. This initializer starts the Queued Transaction Scheduler.
- `class com.arbortext.e3.DiskSpaceChecker` with ID `DiskSpaceChecker`. This initializer starts the Disk Space Checker. It has three parameters:
  - *disable*, which can have the values `yes` or `no` (the default).
  - *interval*, which specifies an interval for checking disk space, expressed in seconds. It is set to 600 (the default).
  - *threshold*, which specifies the threshold for issuing a warning to the servlet log that disk space is low. The value is specified as a series of thresholds where a single specification is `drive_letter:integerCriteria..`  
The wild card `*` can be specified for drive letter, meaning all drives on the Arbortext PE server.

---

The criteria is one of the characters % for percent of available space, K for kilobytes of available space, M for megabytes of available space, G for gigabytes of available space, or T for terabytes of available space.

*threshold* is set to \*:10% (the default), which means issue a warning for any disk with less than 10% available space.

Multiple specifications are separated by ;, for example,

\*:20%;c:100M;g:400G. This example issues a warning if the C drive has less than 100 megabytes available space, the G drive has less than 400 gigabytes available space; and if any other drive on the server has less than 20% available space.

The Arbortext Publishing Engine index page Status and Configuration report has a Disk Space section where you can view the results of the last time disk space was checked.

In addition, if a disk has less space available than the amount specified by *threshold*, the Disk Space Checker writes a WARN level log message into the servlet log (pe1og.xml in Tomcat's log directory). To see the WARN messages from the Disk Space Checker in the log, set the *debug* flag in e3config.xml to true.

# 8

## Requesting Administrative Information

Requesting a Status Report .....	112
Requesting a License Report .....	114
Requesting a Version Report.....	115
Monitoring the Transaction Archive .....	115
Requesting the Queue Reports .....	118
Requesting a Java Properties Report.....	121
Requesting Web Services Definitions.....	121
Requesting a Publishing Configuration Report.....	121
Usage Report.....	123
Requesting a Zip Archive for Troubleshooting.....	123
Requesting a Rescan of Publishing Configuration .....	123
Reloading Scripts .....	123
Running the Samples .....	124

You can request various reports from the Arbortext PE Request Manager to get information on status, license, version, publishing configuration, and Java system properties. All of these reports are available from links on the Arbortext Publishing Engine index page. The index page has links for obtaining reports about Arbortext Publishing Engine environment and activities, converting a demo document to a variety of output formats, running Arbortext Publishing Engine test applications, reloading scripts or publishing configuration information on the server, and obtaining Arbortext Publishing Engine web services (WSDL) information.

After you've successfully installed and configured Arbortext Publishing Engine, this page is available from a web browser by submitting a URL like the following:  
`http://servername:port/e3`

---

## Requesting a Status Report

This report contains a wide variety of information about the Arbortext PE Request Manager and its Arbortext PE sub-processes. This report is available from the **Status** link on the Arbortext Publishing Engine index page. Check this report to review all the setup information about Arbortext Publishing Engine. The information in this report is helpful for troubleshooting. It includes the configuration settings defined in the `e3config.xml` configuration file. You can check stylesheet and publishing configuration caching as well as the set of allowed functions. You can also check the location where Arbortext Publishing Engine stores transaction archives.

The Arbortext Publishing Engine status report contains:

- Information about the Arbortext Publishing Engine installation, including version, build number, binary path, and run-time data
- System information about the Arbortext Publishing Engine environment
- COM server information as set by the Arbortext Publishing Engine Configuration program
- Allowed functions, by type and pattern, as configured in the `e3config.xml` file
- Global parameter settings for temporary file locations and naming conventions, including the installation path, the temporary directory with prefix and suffix values, and the active and archived transaction directories. Values for debugging and temporary file deletion are also reported.
- Configuration summary for each Arbortext PE sub-process pool defined in the `e3config.xml` file, including their parameter values (if the pool is enabled). If the pool is enabled, the configuration summary includes run-time data and whether Request Tests have been configured.
- Configuration summary for each Cache Manager defined in the `e3config.xml` file, as well as associated run-time data. By default, the publishing configuration and stylesheet cache managers are defined.
- Queue manager configuration and run-time data, if available
- Configuration summary for each Request Handler, as implemented and configured in the `e3config.xml` file
- Summary of Request Selector tests and their parameter values as configured in the `e3config.xml` file. Request Selector tests are associated with specific Subprocess Pools (also configured in the `e3config.xml`) as filters for routing requests to a particular pool. The Request Selector tests are listed in the Request Test Criteria section for each pool.



- 
- Configuration summary for Initializers as defined in the `e3config.xml` file and associated run-time data if it's available. By default, the publishing configuration and stylesheet transformations are initialized at startup.
  - A list of Arbortext applications installed on the server, such as the Technical Information Application. These applications are installed in the `application` directory of the Arbortext Publishing Engine installation tree.
  - Transaction archive configuration and status, including location, size, log level, number of transactions, and age of oldest transaction stored.

The following is an example of the Arbortext Publishing Engine Status report, showing the first sections of the report:

## Installation Information

**Version:** Version 6.0.000  
**Date:** November 23, 2010  
**Build:** Build PINE-1327  
**Copyright:** 1999-2010, PTC, Inc.  
**Started at:** 00:35:18 on 11/23/2010  
**Running for:** 38578 seconds (10 hours, 42 minutes, 58 seconds)  
**Path to binaries:** D:\e3\_daily\_build  
**Path to servlet context:** D:\e3\_daily\_build\e3\e3\  
**Subprocess Version:** 6.0 F000  
**Subprocess Build Date:** November 23, 2010  
**Subprocess Build:** PINE-1327

## System Information

**Java Version:** 1.6.0\_17  
**Java Vendor:** Sun Microsystems Inc.  
**Java Folder:** C:\Program Files\Java\jre6  
**Operating System:** Windows 2003  
**Operating System Version:** 5.2  
**Host Architecture:** x86  
**Temporary File Directory:** D:\Apache\Tomcat60\temp  
**Class Path:** D:\Apache\Tomcat60\bin\bootstrap.jar  
**Servlet Container:** Apache Tomcat/6.0.20  
**Servlet API:** 2.5  
**Servlet Context:** Arbortext Publishing Engine

## COM Configuration

Type	Information
COM server status:	present
Default Printer:	Microsoft XPS Document Writer
Deployment Descriptor Location:	
Hard Error Mode:	0
Hidden Desktop Heap Size:	594
IIS Port:	80
Identity:	Launching User
PE Integration with Tomcat:	present
Tomcat Intallation Path:	D:\Apache\Tomcat60
Tomcat Integration with IIS:	present
Tomcat Port:	8080

## Requesting a License Report

The License Information report contains Session Information, including the Service Contract Number, software version and system, and installation path. It provides the number of Processor Cores and Processor Packages, which is for the license on your system.

---

The `License source` provides the information contained in the license environment variable, `PTC_D_LICENSE_FILE`. Use the `License` link to check the license information stored on the system if you are experiencing license problems. See the Arbortext Publishing Engine Licensing chapter of the *Installation Guide for Arbortext Publishing Engine* if you are having license problems.

The license report also provides information about optional components you have installed and whether each is licensed for your site. Use this report to make sure all features for your site are properly licensed and enabled as you expect.

## Requesting a Version Report

This report contains version information for Arbortext PE Request Manager and its Arbortext PE sub-processes, including information about any applications installed in the `application` directory.

This report is available from the **Version** link on the Arbortext Publishing Engine index page. Use this report to verify version and location information about the installation as well as some minimal run-time data.

## Monitoring the Transaction Archive

The Transaction Archive page lists the transactions that have been processed and archived on the Arbortext PE server, according to archiving configuration. This report is available from the **Transaction Archive** link on the Arbortext Publishing Engine index page. The report displays the archived transactions from newest to oldest. The parameters that control the behavior for archiving transactions are described in [The Global Transaction Archive Parameters on page 74](#).

---

### Note

A transaction can be archived in an alternate location that is not available from the Arbortext Publishing Engine index page. The alternate transaction archive allows transactions containing sensitive data to be stored in a location where permission would be required to access them. Refer to [The Global Transaction Archive Parameters on page 74](#) for more information on the alternate transaction archive.

---

Each transaction is identified by a unique **ID** that's assigned at the time the request was received. An ID may be reused only after the transaction it identifies is deleted from the archive. If the archive entries are configured to persist across Arbortext Publishing Engine sessions, IDs of transactions in the archive at startup also won't be reused until they're deleted.

---

A transaction name may appear on the Transaction Archive page if one has been specified by the Arbortext Editor client, a `transaction-name` query parameter (see [Queuing Query Parameters on page 46](#)), or by the global `com.arbortext.e3.defaultTransactionName` parameter (see [The Global Transaction Name Parameter on page 73](#)).

Each transaction ID number is a link to the Transaction detail page displaying information about the selected transaction. Each transaction has a check box that you can select before choosing an action from the menu at the top. A global check box in the title bar (just above the first transaction) selects all transaction entries. Clearing this check box clears all check marks. Each transaction displays information about its operation, start and end time, result, and the requesting client.

For each transaction in the archive, Arbortext Publishing Engine stores a zip file that contains the following information:

- the request, including the function, URI, headers, query parameters, and request body
- the response, including status code, HTTP headers, and response body
- whether a transmission error prevented the response from being returned to the client
- whether the Arbortext PE sub-process terminated while processing the request. If so, it returns the associated termination error file, assertion message, COM error information, or other relevant error information.
- an application log for the request
- the intermediate files associated with the request


The menu offers the following actions:

-  **Delete**

Put a check mark in one or more boxes and then click this button to delete the selected transactions.

-  **Delete Successful**

Click this button to delete all successful transactions, which are those that have a returned status of 200.

-  **Zip Archive**

---

Put a check mark in one or more boxes and then click this button to return a zip archive containing information about each selected transaction.

## The Transaction Detail Page

When you click the link for the Job ID number, the Transaction detail page is returned. It displays more detailed information about the selected transaction. For each transaction, the report contains the following information:

- Summary of the operation, including name of operation, start and end time stamp, result, client, and a log if available.
- Summary of the response, including encoding, locale, state, status, and a response body (with its content-type) if available
- Summary of the intermediate files produced during processing the request
- Summary of the request, including the request body if available, and detailed information about the request query and client.

The menu offers the following actions:

- **Delete**  
Click this button to delete the transaction.
- **Zip Archive**  
Click this button to return a zip archive containing information about this transaction.

## Application Logging and Intermediate Files

A transaction detail entry and a transaction archive file can contain an application log and a number of intermediate files. An application running in an Arbortext PE sub-process can write to the application log when it processes the request and produces its response. The application log, the intermediate files, and a data file describing the intermediate files are stored by the Arbortext PE Request Manager as internal data associated with a transaction. Each time the Arbortext PE Request Manager directs an Arbortext PE sub-process to handle a request, it will indicate whether application logging is enabled. When the Arbortext PE sub-process returns the HTTP response to the Arbortext PE Request Manager, it will also pass the application log and intermediate files.

Each intermediate file produced during processing is saved to disk, with an optional accompanying MIME type (such as `application/zip-archive` or `text/html`) and a description string. The content of the log and the number and content of the intermediate files are specific to the transaction. This information is not returned to the client as part of an HTTP response (as the HTTP protocol doesn't allow it).

---

The application logging parameters that set what is logged and the log levels are described in [The Global Application Logging Parameters on page 80](#).

A custom application can make entries in the application log and keep intermediate files by calling methods in the following packages:

- Java or JavaScript applications call the **E3ApplicationConfig** interface
- VBScript or ACL applications call the **PEAppConfig** package

Refer to the *Programmer's Guide to Arbortext Publishing Engine* for information on using these packages.

## Requesting the Queue Reports

The Queue List displays the queues that have been configured on the Arbortext PE server. This report is available from the **Queue List** link on the Arbortext Publishing Engine index page (described in [Monitoring and Reporting Using a Web Browser on page 21](#)).

The report displays the queues in order of their configuration in the `e3config.xml` file. You can view the list of queues and enable or disable them manually. From this page, there are links to display a queue's configuration, queued transactions, and completed transactions.

The parameters that control the behavior for queues are described in:

- Global queuing parameters described in [The Global Queuing Parameters on page 76](#).
- Queue parameters described in [Specifying Queues on page 93](#).

## The Queue List Page

The **Queue List** link on the Arbortext Publishing Engine index page opens the **Queue List** web page, where you can get information about each configured queue and its transactions:



- **Queue** displays the name of the queue as specified by its ID.
- **Info** link displays the queue's configured parameters and request test set criteria from the `e3config.xml` file. This is a good way to check that the configuration is set up as you expect.
- **Enabled** reports the queue state.
- **Action** lets you **Enable** or **Disable** the queue. Enabling the queue allows its transactions to be processed during its active period. Disabling suppresses processing altogether, although transactions can still be placed on the queue.

- 
- **Active** reports whether the queue currently active, according to its configuration and the current day and time. You can't make a queue active or inactive from this page.
  - **Queued Transactions** displays the number of transactions in the queue. Clicking the numeral displays the **Queued Transaction List** page of the queued transactions waiting to be processed.
  - **Completed Transactions** displays the number of completed transactions for the queue. Clicking the numeral displays the **Completed Transaction List** page of the transactions that are completed or cancelled and waiting for the client to retrieve or delete.



## The Transaction List Page






For the **Queued Transaction List** and **Completed Transaction List**, the report provides the following:

- **ID**  
Click the ID number to view a **Transaction Status** page
- **User**  
Displays the name of the user who submitted the request. This is usually the user's login name.
- **Client**  
Displays the IP address of the client's machine.
- **Actions**  
You can perform the following actions:

-  Delete  
deletes the transaction
-  Download  
allows you to download the transaction results

For the **Queued Transaction List** page only, you can also perform these actions:

-  Hold  
prevents the transaction from executing and persists each time Arbortext Publishing Engine starts.
-  Execute Now  
moves the transaction to be the next in line for execution.

- 
-  Undo Hold  
removes the hold on the transaction
  -  Move to Top  
moves the transaction to the top of the list
  -  Move Up  
moves the transaction up one place in the list
  -  Move Down  
moves the transaction down one place in the list
  -  Move to Bottom  
moves the transaction to the bottom of the list
  - **Operation**  
Displays the type of request.  
For publishing requests from Arbortext Editor, the `ati-operation-type` parameter provides the description. See [Queuing for Arbortext Editor Clients on page 47](#) for information.
  - **Status**  
Displays the current status of the transaction. See [Transaction States on page 39](#) for an explanation.
  - **Submitted**  
Displays the time the transaction was received from the client.
  - **Priority**  
Displays the priority of the request. If no priority was specified, the transaction is assigned a priority of 3 by default.
  - **Started**  
Displays the time the Queued Transaction Scheduler began processing the transaction.
  - **Running**  
Displays the elapsed time since processing began.



---

The **Transaction Status** page provides the same information, but you can also see the time it finished. You can **Retrieve Result** or **Discard Result** in its data file format. A result is whatever is being returned; it can be the published document or a document containing errors and other information.

The form of the result depends on the form requested by the client. For instance, Arbortext Editor clients will receive a data file that is read by the client before extracting the resulting published document. If you click the **Retrieve Result** on that type of transaction, the file will not be a human readable file. For other applications, including the sample applications on the Arbortext Publishing Engine index page, clicking **Retrieve Result** can display the published documents, such as PDF and HTML.

## Requesting a Java Properties Report

This report is available in HTML format from the **Java Properties** link on the Arbortext Publishing Engine index page. This report contains the all the details about the Arbortext Publishing Engine JVM. Check this report to determine whether the Arbortext Publishing Engine Java environment is what you expect.

## Requesting Web Services Definitions

This link returns the WSDL definitions document, which you will need if you are implementing SOAP as your transmission protocol. To enable SOAP, open Arbortext Publishing Engine Configuration and choose the **Tomcat** tab, and choose **Enable** for **SOAP Support**.

This file is available from the **Web Services Definitions** link on the Arbortext Publishing Engine index page. Consult the *Programmer's Guide to Arbortext Publishing Engine* for information on using SOAP with Arbortext Publishing Engine.

## Requesting a Publishing Configuration Report

This report contains publishing configuration used by clients such as Arbortext Editor, including information about the document types and related publishing files installed on the Arbortext PE server that are used to fulfill publishing requests. It also reports Arbortext Import/Export templates, applications installed in the `application` directory, and the versions of Arbortext Editor clients it supports.

---

This report is available in HTML format from the **Short** link on the Arbortext Publishing Engine index page. Use this report to determine whether all your document types, stylesheets, pipeline filters, and other publishing components are available on the Arbortext PE server as expected.

Other forms of this report are also available:

- XML form

**Detailed** link on the Arbortext Publishing Engine index page.

In addition, this report contains check sums and other data that allows Arbortext Editor clients to compare their installations with the Arbortext PE server installation.

- Text form

**Log** link on the Arbortext Publishing Engine index page.

In addition, this report contains information on how Arbortext Publishing Engine built the report by providing document type-specific information such as the directories it scanned, files it found, and files it expected to find that are missing,

## Publishing Management for Arbortext Editor Clients

Arbortext Editor can report Arbortext Publishing Engine configuration information from its **Help ▶ About Arbortext Editor ▶ PE Configuration** menu command. The report can be used to determine duplicate stylesheet names on the server. If a stylesheet name is not unique on the server, Arbortext Publishing Engine uses the first one it finds.

Arbortext Editor can compare its publishing configuration with Arbortext Publishing Engine publishing configuration using its **Tools ▶ Compare Config with PE** menu command. The Publishing Configuration Comparison report notes the differences between the publishing environment on the client and on the server. This report is helpful in troubleshooting client complaints about publishing processing.

If your applications are using a content pipeline for processing large documents and have memory consumption problems, refer to the `doc__estimate_dfs` function and `set bigjobthreshold` command option online help topics (Arbortext Editor or Arbortext Publishing Engine Interactive ) for information on improving content pipeline processing.

---

## Usage Report

You can generate a Usage Report containing information about client usage and transactions. It lists clients, IP addresses, the number of transactions requested by each client, and when the most recent transaction was performed. The data does not persist between sessions and is not saved to disk. You can save the report using the web browser's **File ▶ Save** capability.

## Requesting a Zip Archive for Troubleshooting

You can retrieve a zip archive of application, document type, custom directory, environment, preferences, cache, and configuration information affecting the Arbortext PE sub-processes. This action is available from the **Application Save** link on the Arbortext Publishing Engine index page. You can also run **Tools ▶ Save Application** on the Arbortext Editor client requesting publishing processing from Arbortext Publishing Engine. Use these methods to retrieve the set of information that the PTC technical support staff would need to help troubleshoot a problem you have reported.

You can retrieve a zip archive of everything in the application save archive plus server environment and publishing information from the Arbortext PE server. This action is available from the **All Available Information** link on the Arbortext Publishing Engine index page. Use **All Available Information** to retrieve a larger set of information that the PTC technical support staff would need to help troubleshoot a problem you have reported.

## Requesting a Rescan of Publishing Configuration

This link instructs an Arbortext PE sub-process to scan the publishing configuration and reload the cache used by Arbortext Editor clients. This action is available from the **Rescan Publishing Configuration** link on the Arbortext Publishing Engine index page. Use **Rescan Publishing Configuration** to update the document type information used by Arbortext Editor clients without restarting Arbortext Publishing Engine or its Arbortext PE sub-processes. The **Rescan Publishing Configuration** process will scan the document types available and add any new ones to the list.

## Reloading Scripts

This link instructs Arbortext PE sub-processes to reload custom ACL, JavaScript, and VBScript scripts. This action is available from the **Reload Subprocesses** link on the Arbortext Publishing Engine index page. Use **Reload Subprocesses** to load

---

changes you've made to ACL, JavaScript, and VBScript files without restarting Arbortext Publishing Engine or its Arbortext PE sub-processes. This function reloads ACL, JavaScript, and VBScript programs from the `PE_HOME/custom/init` subdirectory. It also clears the previous stylesheet cache and loads changed stylesheets.

The Arbortext PE sub-process doesn't reload its initialization files until it receives the next request for processing. This feature is especially convenient in the development and testing phase, when you may need to reload your scripts frequently without the need to stop and start Arbortext Publishing Engine.

---

 **Note**

**Reload Subprocesses** (and the `f=init` function) does not affect Java applications. After an Arbortext PE sub-process has started, its JVM can't reload a `.class` or `.jar` file. If you change a Java application, you must stop and restart the servlet container to make the updated Java application available to Arbortext Publishing Engine.

---

## Running the Samples

In the **Test Arbortext Publishing Engine** section, you can convert a sample XML file to the specified formats, all supported by the Arbortext Publishing Engine built-in conversion feature. Consult the *Programmer's Guide to Arbortext Publishing Engine* for information on using it.

---

 **Note**

When you specify the HTML output, graphics will not be displayed in the returned file. This request simply returns the HTML page. Ordinarily, graphics and the document are returned in a zip archive, which can't be displayed in a web browser.

---

You can also import a variety of sample documents into XML. Consult the *Reference Guide to Arbortext Import* and the online help for information on using Arbortext Import.

The PE Test Applications are sample applications in Java, JavaScript, VBScript, and ACL that return an HTML report about the Arbortext PE server environment. The samples are useful to obtain this information, as well as to show how an application is implemented. Refer to the *Programmer's Guide to Arbortext Publishing Engine* for more information.

# 9

## Troubleshooting Arbortext Publishing Engine Operations

Using Arbortext Publishing Engine Interactive for Testing .....	126
Troubleshooting Publishing .....	126
Using the Arbortext Publishing Engine Test Utility .....	127
Troubleshooting Errors .....	127
Getting Trace Information.....	128
Publishing Issues .....	129
Reporting Problems to PTC Technical Support .....	130

Troubleshooting tips cover interactive testing, tracing and diagnostic utilities, log files, and virtual memory out of space error. If you are having trouble with licensing, refer to the Arbortext Publishing Engine Licensing chapter of *Installation Guide for Arbortext Publishing Engine*.

---

## Using Arbortext Publishing Engine Interactive for Testing

Occasionally, you may find it helpful to launch Arbortext Publishing Engine Interactive for testing or troubleshooting purposes. Launch Arbortext Publishing Engine Interactive from its shortcut on your PTC program group.

---

### Note

If you've configured Arbortext Publishing Engine to run as a specified user account, you must log in using that account if you want to run Arbortext Publishing Engine Interactive. When Arbortext Publishing Engine is configured this way, Windows doesn't allow Arbortext Publishing Engine Interactive to run under a different user account for security reasons. For more information about configuring Arbortext Publishing Engine to run as a specified user account, refer to *Installation Guide for Arbortext Publishing Engine*.

---

## Troubleshooting Publishing

When Arbortext Editor is using Arbortext Publishing Engine for publishing documents, you can gather transaction information on the Arbortext PE server from the Arbortext Publishing Engine index page. Refer to [Monitoring the Transaction Archive on page 115](#) for more information.

In a publishing request sent from Arbortext Editor, users can choose transaction names containing Unicode non-ASCII characters. These characters are sent as part of the query parameters contained in the HTTP request. As a result, Tomcat needs to be configured to encode non-ASCII characters using UTF-8.

Edit the Tomcat configuration file `conf/server.xml` and add the attribute `URIEncoding="UTF-8"` to every Connector that is in use (not commented out) in the file.

---

### Note

The `URIEncoding="UTF-8"` attribute is a global setting and affects every servlet running under Tomcat.

---

---

## Troubleshooting on the Arbortext Editor Client

The Arbortext Editor client can also generate additional publishing and logging files useful for troubleshooting by setting `set debugcomposition` to `on` (available in the **Advanced** preferences from **Tools ▶ Preferences**) and then running the **Tools ▶ Save Application**. By default, `set debugcomposition` is turned off.

## Using the Arbortext Publishing Engine Test Utility

The Arbortext Publishing Engine Test Utility is an interactive test tool that you can use to test Java, JavaScript, and Arbortext Command Language (ACL) Arbortext Publishing Engine applications, as well as Arbortext Publishing Engine document conversion parameters.

The Arbortext Publishing Engine Test Utility lets you specify the parameters and their values for custom Arbortext Publishing Engine applications. The utility constructs a query string from these parameters and values, and then validates this string. You can also run tests, and the utility reports the results. If errors occur, they're included in the report.

Once you have created a list of tests, you can save them to disk as an XML file. You can then reload the file, and run all, or some of, the tests. You can launch the Arbortext Publishing Engine Test Utility as a stand-alone program or in embedded mode from the Arbortext Publishing Engine Interactive **Tools** menu. Refer to the *Test Utility User's Guide* for information on using it.

## Troubleshooting Errors

Review the information for how to use the tracing and logging parameters in the `e3config.xml` file, as well as other tracing and logging tips. If you're troubleshooting Arbortext PE sub-process startup problems, you can check the transaction archive.

When you're troubleshooting the Arbortext Publishing Engine, you may decide to run Arbortext Publishing Engine Interactive. If you created a specific user account for Arbortext Publishing Engine, be sure to log on the system with that user account name to better simulate its environment.

You can also run Arbortext Publishing Engine Configuration program from your PTC program group. It reports information about your setup, including COM registration. You can also check your Tomcat integration.

Be sure to check all log and temporary directory folders used by the servlet container and the Arbortext PE server. Some of these may be explicitly set in the `e3config.xml` file. Check that the paths specified there exist on the system.

---

The Tomcat log files in `TOMCAT_HOME\logs` contains log messages written by the Arbortext PE Request Manager. The `pelog.xml` file contains log4j messages that can be used for troubleshooting.

---

 **Note**

- If the Tomcat Windows service won't start, it's possible that it can't find `msvcr71.dll`. Try updating the Windows *PATH* environment variable to add the JRE bin directory. For example:

```
C:\Program files\Java\jre1.1.8_45\bin;
```

Tomcat has an AJP connector defined in `server.xml` which monitors port 8009. However, if port 8009 is already in use, Tomcat doesn't address the conflict. Arbortext Publishing Engine Configuration configures `isapi_redirect.properties` to use port 8009 to integrate with Tomcat.

---

## Getting Trace Information

The **Arbortext Diagnostics** tool is a program available only on Windows that displays tracing information. It tracks identification information, requests and responses, and exceptional conditions. You can launch it from the shortcut called **Arbortext Publishing Engine Diagnostics** in your PTC program group.

When an Arbortext PE sub-process starts, it launches an agent that gathers information from the Arbortext PE Request Manager and its Arbortext PE sub-processes. The agent passes it to the **Arbortext Diagnostics** program which has two tabs:

- **Trace Log** tab displays tracing messages.

Tracing messages about the Arbortext PE Request Manager and Arbortext PE sub-processes are relayed to **Trace Log**.

On the **Arbortext Diagnostics** menu, you can choose **File ► Edit Trace Log** to send the tracing messages to your default text editor. From the text editor, you can save the log to a file. If you are reporting a problem, you should generate and send this file with any other diagnostic information to Technical Support.

- **Module List** tab displays identification information about the Arbortext PE sub-processes as well as:
  - Host machine name and user name
  - Status: running or busy (that is, no response in 1/10 second)
  - Process ID of the Arbortext PE sub-process
  - Build number of the Arbortext PE sub-process executable



- 
- Full path to the binary that was executed

From the **Arbortext Diagnostics** window, you can choose **Options ▶ Preferences** to set the buffer size for the trace log and remember the window's screen position when the program is restarted. You can choose **Options ▶ Always On Top** to keep the **Arbortext Diagnostics** window on top of other application windows. Your choices are saved to the directory where **Arbortext Diagnostics** started.

If **Arbortext Diagnostics** is not running, **Trace Log** information is discarded. After you start **Arbortext Diagnostics**, subsequent tracing information is acquired and displayed.

## Publishing Issues

In a publishing request sent from Arbortext Editor, users can choose transaction names containing Unicode non-ASCII characters. These characters are sent as part of the query parameters contained in the HTTP request. As a result, Tomcat needs to be configured to encode non-ASCII characters using UTF-8.

Edit the Tomcat configuration file `conf/server.xml` and add the attribute `URIEncoding="UTF-8"` to every Connector that is in use (not commented out) in the file.

---

### Note

The `URIEncoding="UTF-8"` attribute is a global setting and affects every servlet running under Tomcat.

---

When publishing very large documents to PDF on Windows, the publishing process may fail with an `Out of virtual memory space` error message. This situation can occur when the publishing process encounters a Windows 2GB memory addressing limitation. You can update your Windows `boot.ini` file to address 3GB of memory using the instructions provided on Microsoft's web site at:

[www.microsoft.com/whdc/system/platform/server/PAE/PAEmem.msp](http://www.microsoft.com/whdc/system/platform/server/PAE/PAEmem.msp)

---

### Caution

Ensure that you make the change exactly as described in the Microsoft documentation. Incorrectly modifying system files can leave your workstation in an unstable state.

---

---

## Reporting Problems to PTC Technical Support

If you are troubleshooting a custom application in consultation with PTC Technical Support, you can save the document, stylesheet, and environment information necessary to reproduce a problem publishing a document. You can run **Tools ► Save Application** on the Arbortext Editor client requesting publishing processing from Arbortext Publishing Engine, or by performing the application save on the server using the **Application Save** link on the Arbortext Publishing Engine HTML page, available from a URL like the following:

```
http://server-name:port/e3/
```

As part of a support call, you may be asked to collect all available information from the server using the **All Available Information** link on the Arbortext Publishing Engine index page.

# 10

## Repository Connection Sample Script

Connecting to a Repository Adapter..... 132

Arbortext Publishing Engine supports a connection to a repository using a script placed in the Arbortext PE server `custom\init` directory. A sample connection script is available from the `e3\samples` directory.

---

## Connecting to a Repository Adapter

JavaScript sample functions provide a starting point for implementing repository connections using Arbortext Publishing Engine. A sample JavaScript function in `PE_HOME\e3\samples\javascript\e3samples.js` shows how to establish a repository connection. The function name is specified in an `f=javascript` HTTP request to Arbortext Publishing Engine. You must update the list of `AllowedFunctions` in the `e3config.xml` configuration file to add the JavaScript function names you want to use. Place your modified JavaScript file in `PE_HOME\custom\init`. The functions in it are automatically loaded when the Arbortext PE sub-process starts.

---

### Note

If repository credentials or other sensitive information is stored in `web.xml` or `e3config.xml`, you should remove permission to access the ACL, JavaScript, VBScript and Java sample applications from the `AllowedFunctions` list in the `e3config.xml` configuration file. These sample applications display the global parameters, which would be a security issue if the parameters contain confidential information.

---

You can give a specific Arbortext Publishing Engine user account exclusive permission to read a file containing user credentials. These sample functions show how to read such a file on the server and pass the credentials. The function can retrieve and pass a valid `username` and `password` to establish the repository connection.

The sample function `repository_connect_windchill` establishes a connection to Windchill PDMLink or Arbortext Content Manager with the PTC Server connection.

If you will be using these samples to initiate a permanent connection to the repository so that Arbortext Publishing Engine operations such as `f=convert` will have access to the repository's objects, the `session.disconnect()` line in the script will need to be removed or commented out in the function.

By default, Arbortext Publishing Engine runs on Windows under a local account called `SYSTEM`. You can create a different user account for Arbortext Publishing Engine (see *Installation Guide for Arbortext Publishing Engine* for instructions). Access to files on a Windows server machine is controlled by NTFS security. You can give this specific Arbortext Publishing Engine user account exclusive permission to read from a particular file.

After configuring an Arbortext Publishing Engine user account, set the permissions on your credentials text file to give exclusive read access to the Arbortext Publishing Engine user account. All other accounts should have no access. To test the Arbortext Publishing Engine user account's access to the secure

---

file, log in to Windows as the Arbortext Publishing Engine user ID and try to access the file. After you've excluded other users with accounts on your system, you can log in using one of those accounts and make certain the file is not accessible.

Using an ASCII text file for the password file prevents someone from trying to obtain access to the file using a HTTP request containing the `f=convert` function. An Arbortext Publishing Engine request to convert and return a text file will fail, even if the request specifies the correct path and file name for the credentials file.



# Index

## A

- applications
  - allowing in e3config.xml file, 107
- Arbortext Publishing Engine
  - configuration
  - testing, 21

## C

- cache manager, 92
- client function
  - allowing in e3config.xml file, 107
- configuration files
  - getting values from, 69
- configuration parameters
  - retrieving the values, 69
- configuring Arbortext Publishing Engine
  - testing, 21
- configuring PE
  - Advanced tab, 63

## F

- file prefix
  - temporary, 83
- file suffix
  - temporary, 83

## I

- initializers, 109

## P

- PE server

- usage report, 123

## R

- request handler, 83
  - Arbortext Publishing Engine, 83
- request selector, 89

## S

- Sub-process location, 82
- Sub-process pool, 100
  - default, 106

## T

- temporary directory, 82
- temporary file prefix, 83
- temporary file suffix, 83
- test sets, 91
- testing Arbortext Publishing Engine
  - configuration
  - using Arbortext Publishing Engine
    - index page, 21

## W

- Windchill Visualization Service, 105