Scaling Retrieval and Classification (TREX)

TREX 6.0 for EP 5.0 SP5

Since TREX 5.0 SP4, SAP has developed TREX so that the components can be implemented in EP 5.0 SP5 and higher as well as in EP 6.0. Since the same TREX version is used for both products, it was decided to call the release 6.0. TREX 6.0 is not a beta version - it meets the same standards of quality as TREX 5.0 SP4.

Copyright

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Typographic Conventions

Type Style	Description
Example text	Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths and options. Cross-references to other documentation.
`Example text`	Emphasized words or phrases in body text, titles of graphics and tables.
EXAMPLE TEXT	Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Screen output. This includes file and directory names and their paths, messages, source code, names of variables and parameters as well as names of installation, upgrade and database tools.
`EXAMPLE TEXT`	Keys on the keyboard, for example, function keys (such as `F2`) or the `ENTER` key.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.

Scaling TREX 8
Overview 9
Components 9
Java Client 10
Web Server with TREX Extension 10
Queue Server 10
Preprocessor 11
Index Server 11
Name server 12
ISAPI Register 12
TREX Demon 12
Distributed TREX System 13
Scenarios 13
A Small Distributed System 14
Error-Tolerant and Scalable System 15
Distribution Options 17
Implementation 17
Name Server 18
Load Distribution 19
Activation 21
Watchdog Function 21
Name Server Replication 22
Index Replication 23
Replication Flow 24
Load Distribution 24
Automatic Start-Up of TREX Services 25
Distributed Installation with Name Server 26
Checklists 26
Implementing a Distributed System from Scratch 27
Activating the Name Server Later On 28
Adding Hosts to a Distributed System 29
Post-Installation Configuration 30
Configuring the TREX Demon 30
Example Configuration for a Small Distributed System 32
Example Configuration for a Large Distributed System 34
Registering the Name Server with All Remaining TREX Servers 38

Checking Performance Settings for the Operating System 39
Enabling Access to the TREX Installation Directory 40
Changing the User for the TREX Service 42
Activating the Watchdog 43
Activating the Watchdog Temporarily 44
Activating the Watchdog Permanently 45
Registering the Name Server with Content Management 46
What happens when the name server goes down? 47
Name Server Replication 48
Implementing Name Server Replication 49
Configuration 50
Configuring the TREX Demon 50
Registering the Name Server with All Remaining TREX Servers 51
Activating the New Name Server 52
Testing Name Server Replication 53
Registering the Name Server with Content Management 55
How does name server replication work? 56
What happens when one of the name servers goes down? 58
Index Replication 60
Implementing Index Replication 61
Configuration 61
Summary 62
Setting the Master as the Default Index Server 64
Configuration of the Master 65
Activating the Python Extension Handler. 65
Activating Automatic Export 66
Sharing the Replication Directory 68
Structure of the Replication Directory 69
Configuration of the Slaves 70
Activating the Python Extension Handler. 70
Activating the Slave Extension 71
Configuring Automatic Import 72
TREXImport.ini Configuration File 73
Example Configuration 76
Configuring the TREX Demon 79
How does index replication work? 81
Deleting and Initializing a Replicated Index 82

Distributed Preprocessing 83
Starting and Stopping TREX 84
Starting and Stopping the TREX Service 85
Starting and Stopping Individual TREX Servers 86
Starting and Stopping the TREX Demon in Debug Mode 88
Starting and Stopping the Web Server 89
Troubleshooting 90
A replicated index could not be deleted from all slaves 90
Frequently Asked Questions 92
General 92
Index Replication 92
Name Server 94
ISAPI Register 95

Scaling TREX

Purpose

Retrieval and Classification (TREX) offers a flexible architecture that allows a distributed installation and modification in line with various scenarios. There are several ways to scale TREX starting from a single TREX server. Scaling allows you to realize the following:

- Load balancing – You can distribute the search and indexing load amongst several servers.
- Reliability – You can ensure the availability of TREX so that another server takes over the tasks of a server that is unavailable.

This document describes how you scale TREX in a portal environment. The name server plays a central role in a scaled environment. Only the use of a name server allows you to implement multiple Web servers and index servers without extensive configuration. Therefore, this document only takes into account scenarios that use a name server.

The target audience of this guide consists of consultants and administrators who want to implement a distributed TREX system.

How To Use This Guide

This guide contains fundamental information on the TREX architecture and distribution. It also includes checklists for the implementation of a distributed system and detailed instructions on all configuration steps that take place after installation.

- The *Overview* section contains fundamental information on the TREX architecture and distribution. Read this information before you begin to implement a distributed system.
- The *Distributed TREX System with Name Server* section explains how you implement a distributed system from scratch as well as how you configure a non-distributed system so that it is prepared for distribution. You also find information on how you extend an existing distributed system by adding further hosts.
- The *Name Server Replication* section explains how you implement multiple name servers and activate name server replication.
- The *Index Replication* section explains how you make an index available on multiple search servers.
- The *Starting and Stopping TREX Services* section explains how you start and stop the TREX services.
- The last two sections contain information on troubleshooting in distributed TREX systems, and a list of frequently asked questions on distribution.

Further Information

In order to implement a distributed system, you need to refer both to this guide and to the TREX installation guide.

Use the *Scaling Retrieval and Classification (TREX)* document as an initial point of access – it gives an overview of the entire implementation flow. This document then refers to the TREX installation guide wherever necessary.

Overview

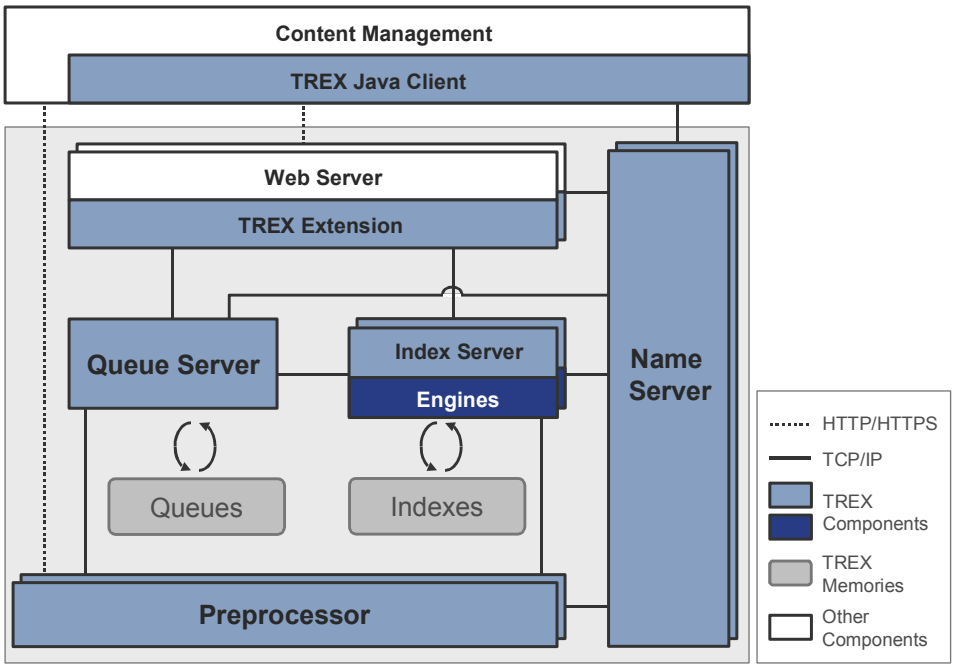
The following sections provide an overview of the central TREX components and introduce the TREX scaling concepts.

Components

TREX includes the following central components:

- Java Client
- Web Server with TREX Extension
- Queue Server
- Preprocessor
- Index Server
- Name Server

The graphic below shows the individual components and how they communicate.



Java Client

TREX provides several interfaces that can be used to integrate TREX functions into an application. The Java client is an interface that Java applications can use to access TREX.

The Java client is integrated into Content Management. This means that the TREX functions are available in Content Management and therefore in the portal.

Web Server with TREX Extension

Content Management (more precisely, the Java client) accesses the TREX functions using a Web server. Communication between Content Management and the Web server takes place using HTTP/HTTPS and XML. The Web server receives requests and forwards them to the index server and queue server. The servers then process the requests.

A TREX component that enhances the Web server with TREX-specific functions is installed on the web server. Technically speaking, this component is an ISAPI server extension for Microsoft Internet Information Server.

Queue Server

The queue server enables the asynchronous indexing of documents. It has a separate queue for each index. It gathers documents to be indexed into one of the queues. It transfers documents to the index server for the actual indexing process at regular intervals. You can use the queue parameters to control when and how many documents are transmitted. This allows you to schedule indexing for times at which the index server does not receive a large amount of search requests.

The queue server forwards the documents to the preprocessor before transmitting them to the index server.

Preprocessor

The preprocessor has two tasks:

- When TREX processes search queries, the preprocessor firstly carries out a linguistic analysis of the search queries. The preprocessor forwards the result of the analysis to the index server, which then carries out the actual processing.
- When TREX receives documents to be indexed, the preprocessor prepares the documents for the actual indexing process. The preparation consists of the following steps:
    - Loading the document

      In the portal, documents are not usually forwarded directly to TREX. Instead, they are forwarded in the form of URIs that reference the storage location of the documents in question. The preprocessor resolves the URIs (that is, it collects the actual documents from the repository).
    - Filtering the documents

      Documents can exist in various formats (Microsoft Word, Microsoft PowerPoint, PDF and so on). The preprocessor filters the documents, that is, it extracts the text content from the documents and then converts them into the UTF-8 Unicode format for further processing.
    - Analyzing the documents linguistically

      The preprocessor uses a lexicon to analyze texts in various languages.

Index Server

The index server is responsible for indexing, classifying, and searching. It receives requests and forwards the, to the TREX engines. The engines provide the actual core functions of TREX. These are:

- The search engine is responsible for standard search functions such as exact, error-tolerant, linguistic, Boolean, and phrase search.
- The Text-mining engine is responsible for classification, searching for similar documents (‘See Also’), the extraction of key words, and so on.
- The attribute engine is responsible for searching in documents properties such as author, creation date, change date, and so on.

Name server

The name server is used with large distributed TREX installations. It uses its databasis to store and coordinate system-wide information. It also ensures that the TREX servers can communicate with each other and that TREX can communicate with Content Management. The name server is also responsible for distributing the system load if more than one TREX server is capable of carrying out a task.

In a distributed scenario, you can install several name servers to ensure that a name server is always available. A replication procedure ensures that the databases of the different name servers are synchronized.

ISAPI Register

An additional TREX component, the ISAPI register, has to run on every Web server in a distributed installation.

The ISAPI register makes sure that the Web server (more precisely, the TREX extension on the Web server) registers itself with the name server when it starts up. The name server then recognizes the Web server and can forward its address on request.

TREX Demon

After the TREX installation, a TREX demon runs on each host. The demon is a central service that starts the actual TREX servers (index server, queue server, and so on) and monitors them during routine operation. If a server becomes unavailable, it is automatically restarted by the demon.

In the standard configuration, the demon starts all servers that have to run for a minimal TREX system. For a distributed installation, you can use the demon’s configuration file to control which servers the demon starts on individual hosts.

Distributed TREX System

There are several ways to scale TREX starting from a minimal TREX system. Scaling allows you to realize the following:

- Load balancing – You can distribute the search and indexing load amongst several servers.
- Reliability – You can ensure the availability of TREX so that another server takes over the tasks of a server that is unavailable.

We recommend that you implement a distributed system if you want to index and manage a large number of documents or think that you will have a large number of search queries.

The TREX Java client is installed with Content Management – not on the TREX servers. Since the description below only refers to the TREX servers, the Java client is not taken into account.

Scenarios

Before installing a distributed TREX system, determine the system requirements. You should take the following factors into account when planning a distributed system:

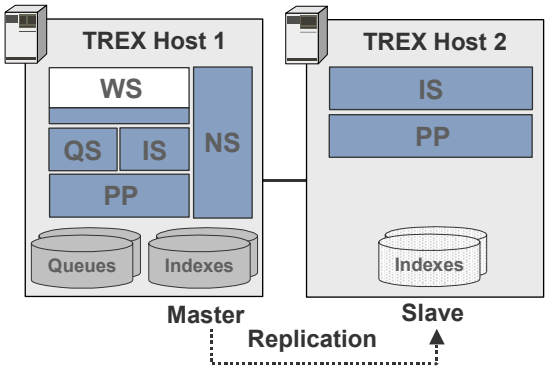
- How many documents need to be indexed?
- What types of documents need to be indexed (Microsoft Word, PDF, HTML, text, and so on)?
- How is the indexing load to be distributed? Will there be a high indexing load only when the initial indexing takes place, or will a large number of documents also be indexed during routine operation?
- How many users will there be? How many parallel search requests will there be?
- What level of availability do you require for TREX?

The answers to these questions determine the number of TREX servers that you require and how the tasks are distributed amongst the servers. Analyze the requirements and planning of the server landscape with an SAP consultant.

The following sections contain an example of a small distributed system and an example of a larger distributed system that is scalable and error-tolerant.

A Small Distributed System

The graphic below depicts a small distributed TREX system with two hosts. The graphic only shows TREX and the portal.



The abbreviations in the graphic are as follows: IS = index server, PP = preprocessor, QS = queue server, and WS = Web server.

In this system, search requests are distributed amongst two hosts. This system has a higher level of performance when searching than does a system that is not distributed.

Tasks for Host 1

Host has two tasks: It is responsible for indexing and classification, and for searching. The host carries out the actual indexing of documents as well as all tasks that take place beforehand: Loading documents from repositories, gathering them into queues, and filtering them and analyzing them linguistically. The preprocessor is responsible for loading, filtering, and analyzing the documents. The queue server gathers the documents into queues and forwards them to the index server according to the configured settings. The index server then carries out the actual indexing of the documents (it inserts the documents into the index). The documents are available for searching as soon as they have been indexed.

The host provides all functions that a non-distributed TREX system would offer. Therefore, all components that would run in a non-distributed system run on this host: The index server, preprocessor, queue server, and Web server. Additionally, a name server runs that ensures communication within TREX and distributes search requests.

Tasks for Host 2

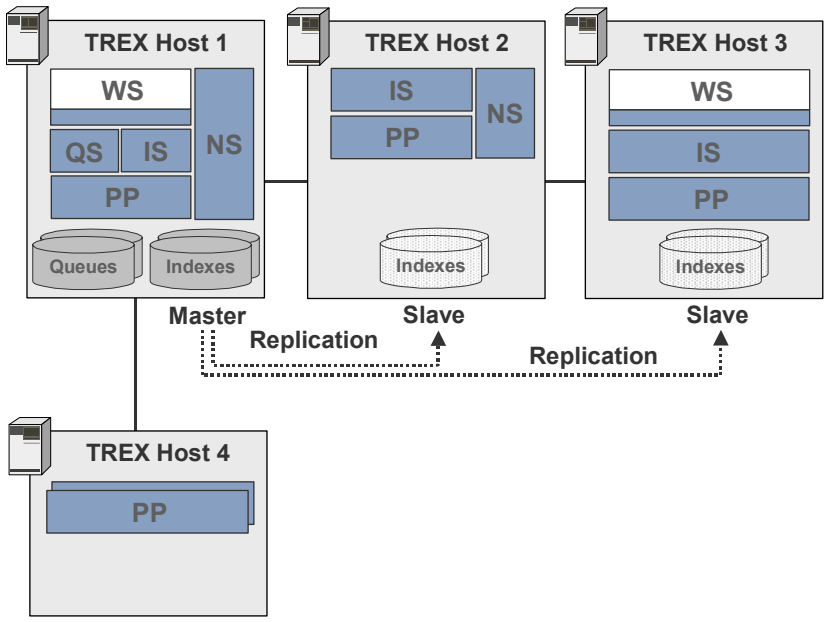
Host 2 is exclusively a search server, that is, it is only responsible for answering search requests. No indexing or classification takes place on host 2.

There are no original indexes on host 2 – it merely contains copies of the indexes stored on host 1. These copies are updated using replication as soon as the original indexes are changed.

Since host 2 is exclusively a search server, only the index server and preprocessor need to run on this host. The preprocessor carries out the linguistic analysis of search requests that are forwarded to host 2. The index server receives the results of the analysis and is then responsible for answering the search requests.

Error-Tolerant and Scalable System

You can further scale a small distributed system and modify TREX to cope with an increased search and indexing load. The graphic below depicts a TREX system with a total of four hosts. As well as increasing performance, this system is aimed at increasing error-tolerance, and reducing single points of failure.



The abbreviations in the graphic are as follows: IS = index server, PP = preprocessor, QS = queue server, and WS = Web server.

Tasks for Hosts 1 and 2

Hosts 1 and 2 carry out more or less the same functions as the two hosts in the small distributed scenario. The only difference is that an additional name server runs on host 2.

In a distributed scenario, the name server has a central role. If the name server goes down, requests are forwarded to a single host and are no longer distributed. To avoid this situation, we recommend that you implement a second instance of the name server when using a larger distributed installation.

Tasks for Host 3

Like host 2, host 3 is exclusively a search server. A second Web server runs on it. This increases the error-tolerance of the system since it avoids a situation where a sole Web server is the single point of failure. For performance reasons, we recommend that you use a second Web server if performance is decreased due to a single Web server receiving too many requests.

Tasks for Host 4

The fourth host is exclusively responsible for preparing documents for indexing. This is why only the preprocessor runs on this host. If host 4 has a higher load when preprocessing than host 1, several instances of the preprocessor may run on host 4.

Since the preprocessing of documents can cause as much system load as the actual indexing, the use of multiple preprocessors can increase data throughput when indexing. We recommend that you use multiple preprocessors if lots of large documents that are not in HTML or text form are to be indexed.

Error-Tolerance

In the depicted distributed system, all TREX components apart from the queue server are installed in multiple. Read access (that is, searching) to this system is error-tolerant because all components involved in the search are installed in multiple.

Write access (that is, indexing) is not currently error-tolerant. If the queue server or index server on host 1 goes down, indexing is interrupted. A further TREX release allows multiple queue servers to be implemented, thereby increasing the error-tolerance of the system.

Distribution Options

The table below provides an overview of the tasks that a host can carry out and the components it needs to do so.

Task	Required TREX Components				
	QS	PP	IS	WS	NS
Collect and preprocess documents	✓	✓			
Preprocess documents		✓			
Index and search for documents		✓	✓		
Search for documents		✓	✓		
Forward application requests to TREX				✓	
Enable communication					✓

The abbreviations in the table are as follows: IS = index server, PP = preprocessor, QS = queue server, NS = name server, and WS = Web server.

Note the following information on a distributed system:

Web Server

The Web server transmits Content Management requests to TREX. The Web server can run on any host including a host on which no other TREX component runs. If the load on a single Web server becomes too great, you can implement additional Web servers on other hosts.

ISAPI Register (Windows Only)

When running a distributed system using Windows, an instance of the ISAPI register component needs to run on each Web server

Name Server

The name server can run on any TREX host. However, if you want to realize a high degree of reliability, you need to set up the name server on a separate host from the other TREX components.

Queue Server

The queue server gathers the documents that are to be indexed and forwards them initially to the preprocessor and then to the index server. At this point, only **one** queue server is beneficial. A further TREX release allows you to run multiple queue servers and distribute the queue server tasks amongst several hosts.

Implementation

When you implement a distributed system, you initially install all TREX software on each host. You then configure the hosts according to the tasks that each host is to carry out.

The most important configuration step is the configuration of the TREX demon. The TREX demon is a central service that starts the actual TREX servers and monitors them during routine operation. When you configure the TREX demon, you define which TREX components are to run on which host. In turn, this determines the tasks of each host.

Name Server

You use a name server in large, distributed TREX installations. The name server contains information on all TREX servers in its databasis. This information includes:

- The type of server – index server, name server, preprocessor, queue server, or Web server with TREX extension.
- The address of the server.
- For an index server, the indexes that are stored there.
- For a queue server, the queues that are stored there.
- For a preprocessor, the services that it offers (load documents using HTTP, GET, filtering, linguistic analysis).

The other TREX servers and Content Management can query the name server databasis and communicate with each other using the information queried.

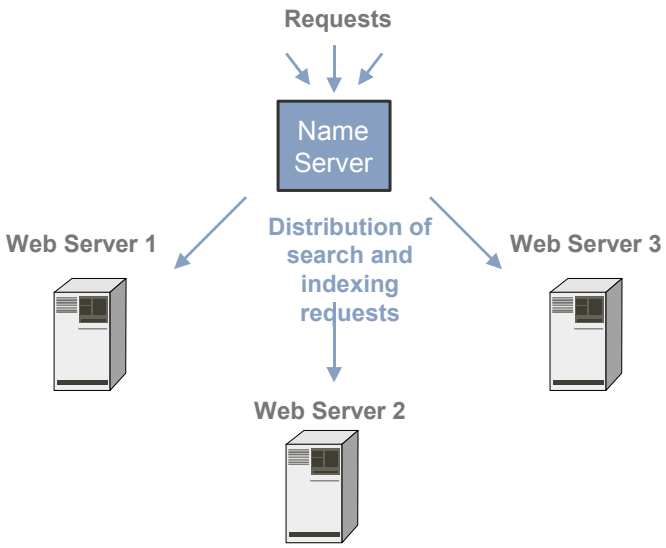
Using a name server is the only way to build up a large distributed server landscape that distributes the load amongst several servers. A name server is the prerequisite for realizing the following:

- Multiple Web servers
- Multiple index servers with automatic index replication

Load Distribution

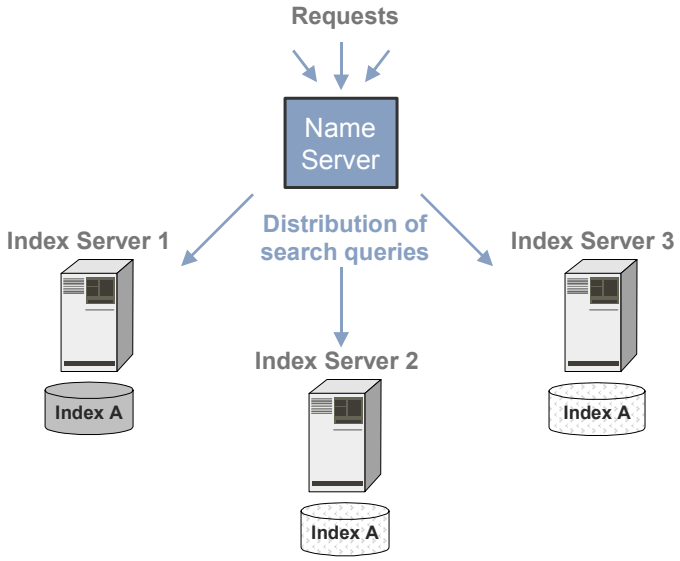
Distributing search and indexing requests amongst Web servers

If the burden on a single Web server becomes too great, you can implement additional Web servers. The name server then distributes search and indexing requests amongst the Web servers being used.



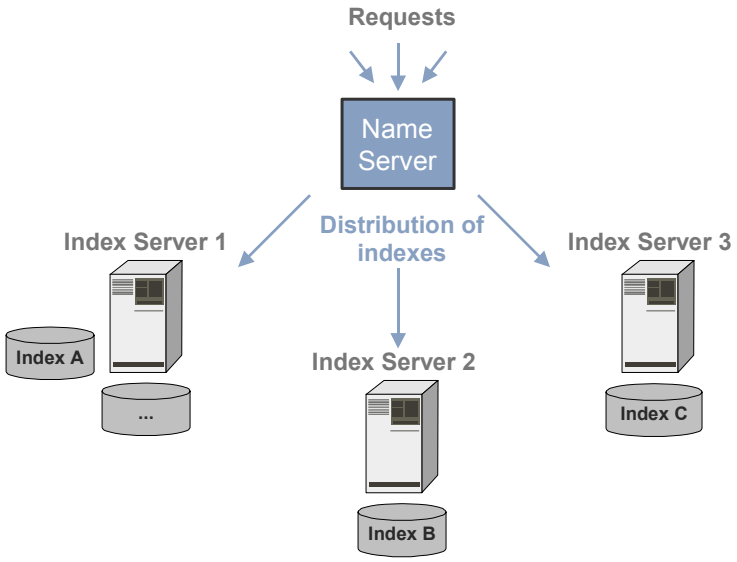
Distributing search queries amongst index servers

You can use index replication to make one index available on more than one server. This ensures that indexes that are used frequently are available for searching, and distributes the search load amongst several servers. The name server keeps information about the server on which an index is stored in its databasis, and then distributes the search queries.



Distributing indexes

If you are using multiple index servers, by default, the name server makes sure that the creation of indexes is distributed amongst the index servers.



The databasis of the name server contains information on which index is stored on which server. Existing indexes are recognized system-wide. The name server makes sure that search and indexing requests are forwarded to the appropriate server.

By default, the name server distributes requests to all index servers registered with it. In a system with index replication, the slaves should normally contain copies of indexes, but no original indexes. You can configure the name server appropriately by indicating that the master server is the default index server. The name server then only forwards create index requests to the default index server. If no index server is indicated as the default index server, all index servers are treated as default.

How does the name server realize load distribution?

The name server uses a round robin procedure that distributes requests to the connected servers in turn.

The name server has a counter for each server. This counter specifies the number of times that the name server has forwarded the address of the server. If the name server receives a request that could be answered by more than one server, it returns the server with the lowest count. The counter for the server in question is then increased appropriately.

Activation

Since the name server is not required for a non-distributed TREX system, it is deactivated by default. In order to implement the name server, you have to configure TREX accordingly, thereby activating the name server. There are two levels of activation:

- TREX-side
- Content Management-side

The distribution scenario that you are implementing dictates whether you need to activate the name server only on TREX-side or also on Content Management-side.

Activation on TREX-Side

You always activate the name server on TREX-side. As soon as you have activated the name server, the TREX servers communicate using it.

When a TREX server starts, it reports to the name server and publishes information about itself. The name server stores this information in its databasis and forwards it as appropriate to querying components.

When a TREX server stops, it informs the name server that it is no longer available. The name server then marks the TREX server in question as inactive. The TREX server and any indexes stored on it are still recognized system-wide, but are not available for use.

Activation on Content Management-Side

If you activate the name server on Content Management-side, Content Management (and therefore the portal) communicates with TREX using the name server. In this case, Content Management always asks the name server for the address of a Web server before forwarding the HTTP request to the Web server in question.

It is only beneficial to activate the name server on Content Management-side if you are implementing multiple Web servers. This is the only way of distributing HTTP requests amongst multiple Web servers.

Watchdog Function

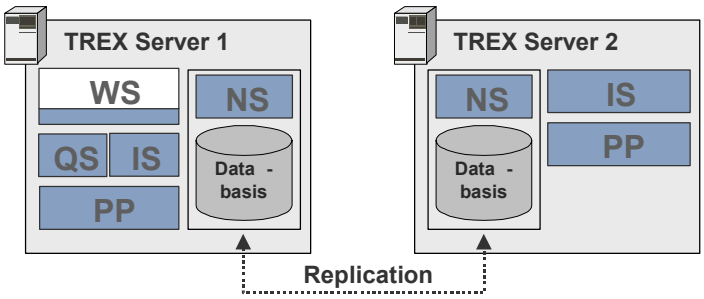
The name server offers a watchdog function that monitors all active TREX servers. The watchdog regularly checks whether the active TREX servers can be reached. If a server does not respond, the name server marks it in its databasis as inactive. This prevents the name server from forwarding the address of servers that are not available.

The watchdog only checks active TREX servers. It does not check whether inactive servers can be reached again. This is not necessary, since when an unavailable server is restarted, it reports to the name server and is recognized as being active again.

Name Server Replication

The name server has a central function in a distributed scenario. It is responsible for load distribution and enables communication between the TREX servers. If the name server is also activated on Content Management-side, it is also responsible for communication between TREX and Content Management.

You can implement multiple name servers for reliability purposes. This allows you to ensure the availability of the name server during routine operation, and prevents a sole name server from being a single point of failure.



The abbreviations in the graphic are as follows: IS = index server, PP = preprocessor, QS = queue server, and WS = Web server.

Normally, all communication takes place using **one** name server. If this name server goes down, another name server can immediately take on the tasks of the unavailable server. A replication procedure makes sure that all name servers have the same information in their databases. This means that whenever a name server goes down, the system can make a seamless change to another name server.

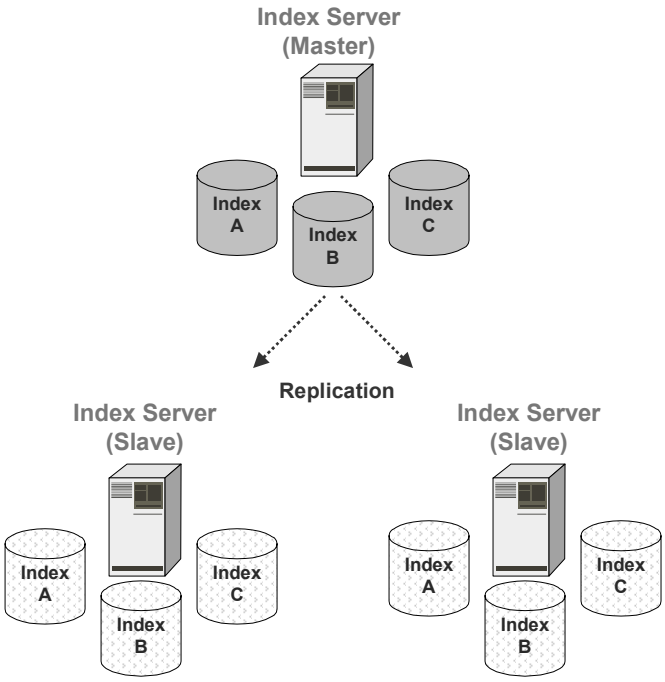
Index Replication

TREX allows you to replicate indexes automatically, thereby making the same index available on multiple servers. This is recommended in a system environment that has a large number of indexes and applications that are searching in parallel.

Index replication offers the following advantages:

- It increases search and text-mining performance. If an index server has to answer too many parallel search requests, the server response time can become too great. If this is the case, you can scale the affected index server by replicating its indexes on additional servers, thereby distributing the search load amongst several servers.
- The availability of indexes for searching is ensured. If an index server service on a particular host goes down, or if the host itself goes down, the remaining index servers can still be used for the search.

When index replication takes place, one index server (the master) retains the original index. All other index servers (the slaves) hold copies of the index and can be used as search servers.



Replication Flow

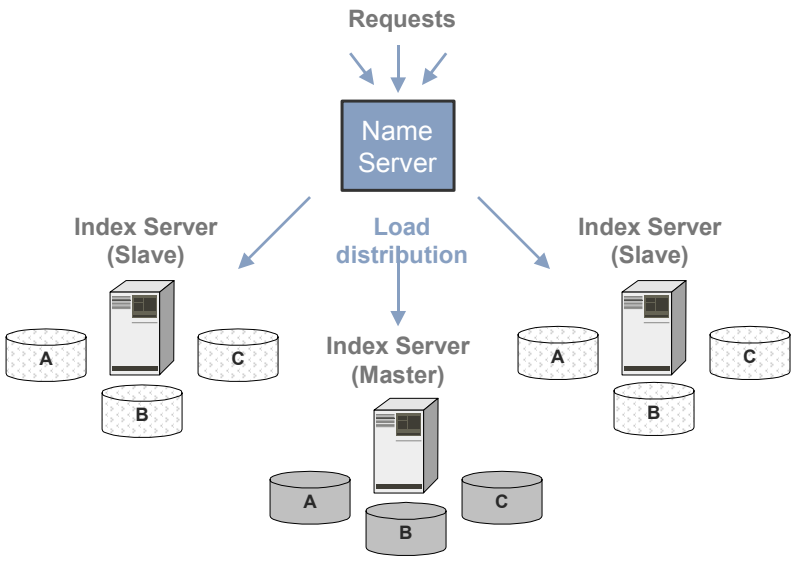
All changes are made to the original index on the master. The replication procedure makes sure that the original index is exported and that the copies are imported by the slaves.

- A Python extension that creates the backups of the original index runs on the master. The export takes place as soon as changes to the original index have been completed.
- On import demon that checks regularly for new index versions runs on each slave. If a new version exists and the time for the next import has been reached, the demon imports the new version.

The indexes can be replicated during routine operation without the need to stop the index servers.

Load Distribution

If an index is available on several servers, the search load is distributed amongst the servers in question. The name server is responsible for this distribution.



See also:

Name Server [Page 18]

Automatic Start-Up of TREX Services

A TREX demon runs on every TREX host. The demon starts the actual TREX services (index server, queue server, and so on) and monitors them during routine operation. If a service goes down, it is restarted by the demon. This ensures that the unavailable TREX services are restarted automatically.

If you are running TREX on Windows, the TREX demon is registered as a service.

Distributed Installation with Name Server

Purpose

The following sections explain how you implement a distributed TREX system with a name server. They contain information on

- How to implement a distributed system from scratch
- How to extend a minimal (that is, non-distributed) system by adding a name server, thereby preparing it for a distributed scenario
- How to add additional hosts to a distributed system

The TREX installation process consists of planning, preparation, installation, and post installation phases. The planning, preparation, and installation phases are the same for both minimal and distributed systems. The configuration steps that take place after the installation are different depending on the system being used.

This document contains checklists that provide an overview of the actions that are required and the sequence in which you carry them out. There are also detailed instructions for all configuration steps after the installation.

For detailed instructions on the planning, preparation, and installation phases, see the TREX installation guide.

Checklists

Purpose

Use the checklists below when implementing a distributed system. Use the links to general descriptions of actions and additional information that supports you when you carry out the actions in question. This ensures that you will not overlook important information.

We recommend that you proceed as follows when implementing a distributed system:

Print out the table below.

Keep to the sequence specified in the table.

- When an implementation step is required, follow the link to the relevant section.
- The carry out the step described in the section in question.
- When you have successfully completed the implementation step, place a checkmark (!) next to the corresponding entry in the table. This allows you to log your progress.
- Then continue with the next step described in the table.

Implementing a Distributed System from Scratch

Purpose

Use the checklist below if you want to implement a distributed system with multiple hosts from scratch. Unless otherwise specified, you carry out each action on all TREX hosts being used.

Planning, Preparation, and Installation

Use the checklists in the TREX installation guide for this phase.

Post Installation

✓	Action
	If the TREX demon is running, stop it (see Starting and Stopping the TREX Service [Page 85]).
	Configure the TREX demon so that it only starts the required TREX servers (see Configuring the TREX Demon [Page 30]).
	Register the name server with all remaining TREX servers [Page 38].
	Check the performance settings of the operating system [Page 39].
	Ensure access to the TREX installation directory [Page 40].
	Change the user for the TREX service [Page 42].
	Start the TREX demon on the individual hosts in the following order: First on the name server Then on all other hosts (See Starting and Stopping the TREX Service [Page 85]).
	Only on the name server: Activate the watchdog [Page 43]
	Only on the portal server, and only if implementing multiple Web servers for TREX: • Register the name server with Content Management [Page 46] • Restart the servlet engine

If you want to implement index replication, additional configuration steps are necessary. If you only want new indexes to be created on the master, you have to indicate that the master is the default index server (see Setting the Master as the Default Index Server [Page 64]). Carry out this configuration step before you create any indexes. Otherwise, new indexes will be created on the slaves as well as on the master.

For more information on index replication, see Index Replication [Page 60].

Activating the Name Server Later On

Purpose

Use the checklist below if you want to extend a non-distributed system by adding a name server. This prepares the system for a distributed scenario. You can add further hosts once you have activated the name server on your existing TREX host.

Prerequisites

You are using **one** TREX host and want to activate the name server on this host.

Configuration

✓	**Action**
	Stop the TREX demon (See Starting and Stopping the TREX Service [Page 85]).
	Configure the TREX demon so that it also starts the following servers: • Name server • ISAPI register (See Configuring the TREX Demon [Page 30]).
	Register the name server with all remaining TREX servers [Page 38].
	Start the TREX demon (See Starting and Stopping the TREX Service [Page 85]).
	Activate the watchdog [Page 43].
	Only on the portal server, and only if implementing multiple Web servers for TREX: • Register the name server with Content Management [Page 46]. • Restart the servlet engine

Adding Hosts to a Distributed System

Purpose

If you have already activated the name server, you can extend the TREX server landscape fairly easily. Use the checklist below when adding hosts. Unless otherwise specified, you carry out each action on all new TREX hosts.

Planning, Preparation, and Installation

Use the checklists in the TREX installation guide for this phase.

Post Installation

✓	**Action**
	If the TREX demon is running, stop it (see Starting and Stopping the TREX Service [Page 85]).
	Configure the TREX demon so that it only starts the required TREX servers (see Configuring the TREX Demon [Page 30]).
	Register the name server with all remaining TREX servers [Page 38].
	Check the performance settings of the operating system [Page 39].
	Ensure access to the TREX installation directory [Page 40].
	Change the user for the TREX service [Page 42].
	Start the TREX demon (See Starting and Stopping the TREX Service [Page 85]).
	Only on the portal server, and only if you are implementing multiple Web servers for TREX and have not yet registered the name server with Content Management. • Register the name server with Content Management [Page 46]. • Restart the servlet engine

Post-Installation Configuration

Purpose

The following sections contain detailed information on the individual configuration steps that may be necessary after the installation. Use the checklists [Page 26] to check which configuration steps need to be made.

Configuring the TREX Demon

Use

The TREX demon is a central service that starts the actual TREX servers (index server, queue server, and so on) and monitors them during routine operation.

The `<TREX_Directory>\TREXDaemon.ini` configuration file defines which servers are started by the demon. In a distributed installation, you modify the configuration file on each host so that the demon only starts the servers that are supposed to run on the host in question.

Prerequisites

The TREX demon is stopped (see Starting and Stopping the TREX Service [Page 85]).

Procedure

1. Open the `<TREX_Directory>\TREXDaemon.ini` configuration file with a text editor.
2. Create new sections for the servers that are not yet entered in the configuration file. A server section is structured as follows:

Parameter	Description
`[<sectionname>]`	Name of the section. The name can consist of letters and numbers but no special characters. Apart from this restriction, you can choose the name freely. Example: `nameserver`, `indexserver`, and so on.
`executable=<program>`	Name of the program to be started. The program names of the TREX servers are: `TREXIndexServer`, `TREXISAPIRegister`, `TREXNameServer`, `TREXPreprocessor`, and `TREXQueueServer`.
`arguments=<arguments>`	Arguments for the program call. • Enter `-i` in `TREXISAPIRegister`. • If several preprocessor instances are to run on the same host, enter the port of the relevant instance. • As a rule, no arguments are needed for the other TREX servers.

Parameter	Description
`startdir=<program-directory`	Directory from which the program is called. As a rule, enter the TREX installation directory here.
`instances=1`	Number of server instances that are to be started. Possible value: 1.  If multiple preprocessor instances are to run on the host, do not change the value of `instances`. Instead, create a separate section for each instance, and enter the port on which the instance in question is to run as the argument.

3. In the `[daemon]` section, under the `programs` parameter, enter the programs that the demon should start. Use the names of the sections that you created. Make the entries in any order.

   ```
   [daemon]
   programs=nameserver, indexserver, ...
   ```

4. Save the file and close the text editor.

See also:

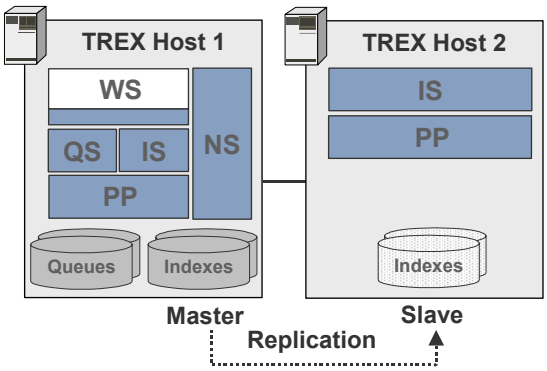
Example Configuration for a Small Distributed System [Page 32]

Example Configuration for a Large Distributed System [Page 34]

Example Configuration for a Small Distributed System

The graphic below displays a small distributed scenario with two hosts. For an explanation of these examples, see Small Distributed System [Page 14]). The graphic shows which TREX servers run on each host.

If you are running the system on Windows, an instance of the ISAPI register must run on host 1 (not depicted in the graphic).



The abbreviations in the graphic are as follows: IS = index server, PP = preprocessor, QS = queue server, and WS = Web server.

The following sections depict the relevant parts of the `TREXDaemon.ini` configuration file for all hosts. The differences from the standard configuration are in bold type.

TREXDaemon.ini on Host 1

```
[daemon]
programs = nameserver, isapiregister, indexserver, queueserver, preprocessor

[nameserver]
executable=TREXNameServer
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1

[isapiregister]
executable=TREXISAPIRegister
arguments=-i
startdir=D:\my_trex_path\SAP\TREX_6
instances=1
```

```
[indexserver]
executable=TREXIndexServer
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1

[queueserver]
executable=TREXQueueServer
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1

[preprocessor]
executable=TREXPreprocessor
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1
```

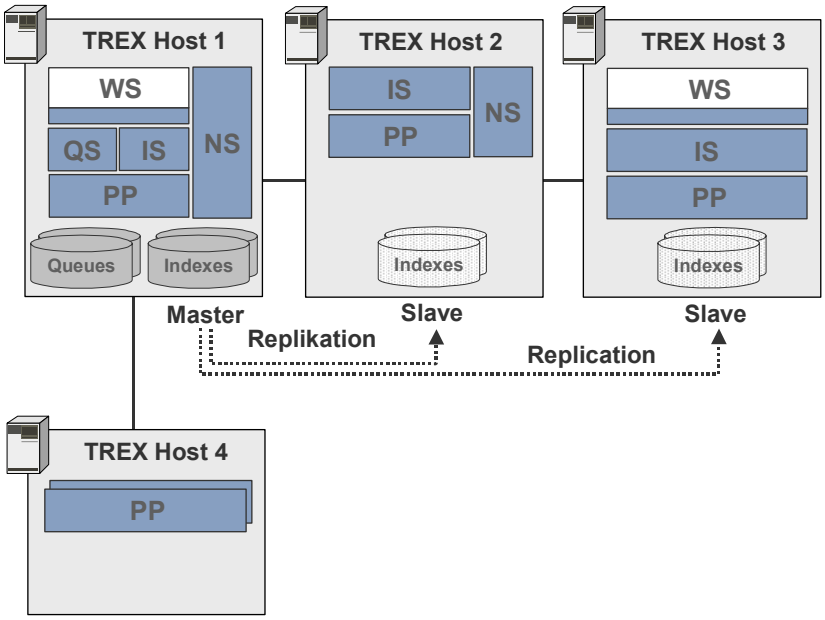
TREXDaemon.ini on Host 2

```
[daemon]
programs = indexserver, preprocessor

[indexserver]
executable=TREXIndexServer
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1

[preprocessor]
executable=TREXPreprocessor
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1
```

Example Configuration for a Large Distributed System

The graphic below displays a large distributed scenario. For an explanation of these examples, see Error-Tolerant and Scalable System [Page 15]). The graphic shows which TREX servers run on each host.

If you are running the system on Windows, an instance of the ISAPI register must run on hosts 1 and 3 (not depicted in the graphic).



The abbreviations in the graphic are as follows: IS = index server, PP = preprocessor, QS = queue server, and WS = Web server.

The following sections depict the relevant parts of the `TREXDaemon.ini` configuration file for all hosts. The differences from the standard configuration are in bold type.

TREXDaemon.ini on Host 1

```
[daemon]
programs = nameserver, isapiregister, indexserver, queueserver, preprocessor


[nameserver]
executable=TREXNameServer
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1
```

```
[isapiregister]
executable=TREXISAPIRegister
arguments=-i
startdir=D:\my_trex_path\SAP\TREX_6
instances=1

[indexserver]
executable=TREXIndexServer
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1

[queueserver]
executable=TREXQueueServer
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1

[preprocessor]
executable=TREXPreprocessor
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1
```

TREXDaemon.ini on Host 2

```
[daemon]
programs = nameserver, indexserver, preprocessor

[nameserver]
executable=TREXNameServer
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1

[indexserver]
executable=TREXIndexServer
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1
```

```
[preprocessor]
executable=TREXPreprocessor
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1
```

TREXDaemon.ini on Host 3

```
[daemon]
programs = isapiregister, indexserver, preprocessor

[isapiregister]
executable=TREXISAPIRegister
arguments=-i
startdir=D:\my_trex_path\SAP\TREX_6
instances=1

[indexserver]
executable=TREXIndexServer
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1

[preprocessor]
executable=TREXPreprocessor
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1
```

TREXDaemon.ini on Host 4

```
[daemon]
programs = preprocessor1, preprocessor2

[preprocessor1]
executable=TREXPreprocessor
arguments=-port 8357
startdir=D:\my_trex_path\SAP\TREX_6
instances=1

[preprocessor2]
executable=TREXPreprocessor
arguments=-port 8359
startdir=D:\my_trex_path\SAP\TREX_6
instances=1
```

The two preprocessor instances need to run on separate ports. You can choose any free ports for this. In the example, the first instance runs on the standard port 8357, and the second instance runs on port 8359. You do not need to specify the standard port, so the `[preprocessor1]` section may also appear thus:

```
[preprocessor1]
executable=TREXPreprocessor
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1
```

Registering the Name Server with All Remaining TREX Servers

Use

To register the name server with all remaining TREX servers, modify the `TrexConfigMgr.ini` configuration file on each TREX host. As soon as a name server is defined in this configuration file, communication at TREX-level takes place using the name server in question.

The configuration files of the individual TREX servers also define whether these servers query the name server for the addresses of other TREX servers.

For example, by default the index server is configured so that it communicates with the local preprocessor and does not query the address from the name server. This ensures that the index server forwards the search queries only to the local preprocessor for linguistic analysis and not to a preprocessor on another host. As a rule, the default settings of the individual servers are correct, that is, you do not need to make changes.

Prerequisites

The TREX demon is stopped (see Starting and Stopping the TREX Service [Page 85]).

Procedure

1. Open the `<TREX_Directory>\TREXConfigMgr.ini` configuration file with a text editor.
2. In the section `[all]`, under the parameter `PNS-address`, enter the address of the name server:

   ```
   PNS-address = tcpip://<hostname>:<port>
   ```

   Unless you specified another port during the installation of TREX, the port is 8355. If the line begins with the hash character #, delete this character.

   ```
   [all]
   PNS-address = tcpip://mytrexhost1:8355
   ```

3. Save the file and close the text editor.

Checking Performance Settings for the Operating System

Use

To optimize the performance of TREX when using the released Windows platform, you need to check your Windows configuration and make changes if necessary.

Optimizing Data Throughput For Network Applications

The Windows installation normally makes caching settings that are optimized for file servers. The operating system then reserves a large part of the main memory for the caching of files. Since this file-system cache impairs performance when indexing, you ought to change these settings.

1. Use the secondary mouse button to choose *My Network Places* from the Windows desktop, and choose *Properties*.
2. Use the secondary mouse button to click on *Local Area Connection,* and then choose *Properties.*
3. Under *Components checked are used by this connection*, choose *File and Printer Sharing for Microsoft Networks*.
4. Choose *Properties*, and select *Maximize data throughput for network applications*.
5. Choose *OK* twice.

Optimizing Performance for Background Processes

Programs such as Microsoft SQL Server and Microsoft Exchange make the setting described below automatically when they are installed. If you have installed one of these programs, you do not need to make any changes.

The setting is only relevant if TREX is running as a service.

1. Use the secondary mouse button on *My Computer*, and choose *Properties*.
2. Choose the *Advanced* tab, and then choose *Performance Options*.
3. Under *Application Response,* choose the *Background Services* field.
4. Choose *OK* twice.

Enabling Access to the TREX Installation Directory

Use

The TREX setup creates the Web site `SAP_TREX` on the Web server and automatically determines a user that is used to access the Web site anonymously. This user needs to have full control permission for the TREX installation directory. There are the following ways of ensuring this:

- Variant 1: You determine the user that is used to access the Web site anonymously. You then give this user full control access for the TREX installation directory.
- Variant 2: You change the access permissions for the TREX installation directory so that all users have full control access.
- Variant 3: You change the user that is used to access the Web site anonymously. Instead of using the default, you enter a local user that has full control access for the TREX installation directory.

Variant 1: Giving the Anonymous User Full Control Access

Determining the user for anonymous access

1. Choose *Start* → *Programs* → *Administrative Tools* → *Internet Services Manager*.
2. Use the secondary mouse button to click on the `SAP_TREX` Web site.
3. Choose *Properties* from the context menu, and then choose the *Directory Security* tab.
4. In the *Anonymous access and authentication control* area, choose *Edit*.
5. In the *Anonymous access* area, choose *Edit*.
6. Select the name that is entered in the *Username* field, and copy it using CTRL+C.
7. Close the Internet Services Manager.

You can then give this user full control access to the TREX installation directory.

Giving full access control to the determined user

1. Go to the TREX installation directory, usually `C:\Program Files\SAP\TREX_6`.
2. Use the secondary mouse button to click on the installation directory.
3. Choose *Properties* from the context menu, and then choose the *Security* tab.
4. Choose *Add*.
5. Use CTRL+V to insert the name of the user that you just copied.
6. Choose *OK*.
7. Select the user, and give it *Full Control* access permission.
8. Choose *OK*.

Variant 2: Giving all Users Full Control Access

1. Go to the TREX installation directory, usually `C:\Program Files\SAP\TREX_6`.
2. Use the secondary mouse button to click on the installation directory.
3. Choose *Properties* from the context menu, and then choose the *Security* tab.
4. Select *Everyone*, and give them *Full Control* access permission.
5. Choose *OK*.

Variant 3: Entering a Local User with Full Control Access as the Anonymous User

1. Choose *Start* → *Programs* → *Administrative Tools* → *Internet Services Manager*.
2. Use the secondary mouse button to click on the `SAP_TREX` Web site.
3. Choose *Properties* from the context menu, and then choose the *Directory Security* tab.
4. In the *Anonymous access and authentication control* area, choose *Edit*.
5. In the *Anonymous access* area, choose *Edit*.
6. In the *Username* field, enter a **local** user that has *Full Control* access to the TREX installation directory.
7. Choose *OK* and close the Internet Services Manager.

Changing the User for the TREX Service

Use

The TREX service needs to run with a user that has access permissions for the network (read- and write-access) so that TREX can load documents from file servers, for example. After the TREX installation, you have to change the user that the service uses to log on. The `LocalSystem` user is used by default. It has no access permissions for the network.

Prerequisites

A user has been set up that has read and write access for the network.

Procedure

1. Choose *Start* → *Settings* → *Control Panel*. Then choose *Administrative Tools* → *Services*.
2. Use the secondary mouse button to click on `TREX Service`.
3. Choose *Properties* from the context menu.
4. Choose the *Log On* tab.
5. Select *This account*.
6. Enter the name of the user and give the password twice.
7. Choose *OK*.

Result

You have to stop and then restart the TREX service before the changes take effect.

Activating the Watchdog

Purpose

The databasis of the name server contains information on which TREX servers are active and which are inactive. When the name server receives requests, it only forwards the addresses of active servers.

When a TREX server starts or stops, it reports the action to the name server. The name server then marks the server in question as active inactive or inactive as appropriate. The name server has a watchdog function for monitoring the active servers. The watchdog can monitor the following servers:

- Index server
- Queue server
- Web server (that is, TREX extension on the Web server)

The watchdog regularly checks whether the active servers can be reached. If a server does not respond, the name server marks it as inactive. The name server does not forward addresses of unavailable servers. As soon as a server that has been marked as inactive is restarted, it reports to the name server and is marked as active again.

The watchdog is part of the name server, and only runs if the name server itself is running. The watchdog is deactivated by default. You can activate the watchdog temporarily or permanently using name server administration.

- If you activate the watchdog temporarily, it is deactivated when the name server is next restarted.
- If you activate the watchdog permanently, it is automatically started when the name server is restarted.

  We recommend that you activate the watchdog permanently. This prevents search queries being forwarded to TREX servers that are not available, thereby causing the search to fail in the portal.

If you are using name server replication, you only have to activate the watchdog on one name server. The replication procedure makes sure that the watchdog is also activated on the other name servers.

Activating the Watchdog Temporarily

Prerequisites

The name server is running (that is, the TREX demon has started).

Procedure

1. Start name server administration on the name server:

   ```
   <TREX_Directory>\TREXNameServerGUI.py
   ```

2. Choose *Change address* and enter the address of the name server.

   ```
   tcpip://<hostname>:<port>
   ```

   Example: **`tcpip://mytrexhost1:8355`**

3. Choose *watchDog.*
4. Enter the value **`start`** in the *action* field.
5. Choose *SEND.*

Result

The watchdog has been started temporarily and checks whether the active TREX servers can be reached every ten seconds. The watchdog is deactivated again when the name server is restarted.

Activating the Watchdog Permanently

Prerequisites

The name server is running (that is, the TREX demon has started).

Procedure

1. Start name server administration on the name server:

   `<TREX_Directory>\TREXNameServerGUI.py`

2. Choose *Change address* and enter the address of the name server.

   `tcpip://<hostname>:<port>`

   Example: **`tcpip://mytrexhost1:8355`**

3. Choose *putServData* and enter the following data:

Field	Entry
service	**`nameserver`**
address	**`__ME__`**
data	**`[_watchdog]`** **`isActive=true`** **`interval=10`**

   The parameter specifies the intervals in seconds at which the watchdog checks the availability of the servers. This is normally every ten seconds.

   The parameter `__ME__` has two underscores before ME and two underscores after it.

4. Choose *SEND.*
5. Then choose *watchDog.*
6. Enter the value **`start`** in the *action* field.
7. Choose *SEND.*

Result

The watchdog is now started and is restarted automatically when the name server is restarted.

Registering the Name Server with Content Management

Use

If you are implementing multiple Web severs, you have to register the address of the name server with Content Management (and thereby the portal), and activate the name server on Content Management-side. Only then are requests distributed amongst the available Web servers.

Prerequisites

The portal and Content Management are installed and running. You can log onto the portal and have the role *KM Admin.*

Procedure

1. Log on to the portal and choose *KM Admin* → *Configuration* → *TREX* → *TREX Java Client* from the top-level navigation bar.
2. Choose *Name Server.* Edit the *nameserver* entry as follows:

Parameter	Entry
`Address`	Enter the address of the name server in the following format: `tcpip://<hostname>.<domain>:<port>` Unless you have specified another port during the installation of TREX, the port number is 8355. Example: **`tcpip://mytrexhost1.mydomain.com:8355`**
`Backup Servers`	Adopt the default setting. This parameter is only relevant if you are using name server replication. For information on name server replication, see Name Server Replication [Page 48].
`Active`	Select this field. This activates the name server in the portal.
`Access Count`	Adopt the default setting.

No other changes are necessary in the portal. You do not need to change the entries under `HTTP Server`, `Default HTTP Server`, `Index Server` and `Queue Server`. The name server updates these settings as soon as you have activated the name server in the portal.

Result

Restart the servlet engine and the portal server so that the changes take effect.

What happens when the name server goes down?

If the name server is down, no load distribution takes place. However, the TREX servers and Content Management can continue to communicate with each other.

- The TREX servers communicate using the information stored in their configuration files.
- If Content Management uses the name server, it continually updates its configuration using information received from the name server. If the name server is down, Content Management uses the information it received most recently.

The TREX servers and Content Management check regularly to see whether the name server is available again. If the name server is available, communication takes place using it.

We recommend that you implement name server replication. This ensures the availability of a name server.

Name Server Replication

Purpose

The name server has a central function in a distributed scenario. It is responsible for load distribution and enables communication between the TREX servers. If the name server is also activated on Content Management-side, it is also responsible for communication between TREX and Content Management.

You can implement multiple name servers for reliability purposes. This allows you to ensure the availability of the name server during routine operation, and prevents a sole name server from being a single point of failure. A replication procedure makes sure that all name servers have the same information in their databases.

The following sections describe how you set up name server replication and how name server replication works. They also contain information on what happens if a name server goes down.

You can activate the additional name server on one of the existing TREX hosts. We only recommend that you set up the name server on its own host if you want a very high degree of reliability.

Implementing Name Server Replication

Purpose

Use the checklist below for the implementation of name server replication/ Keep to the sequence specified in the table. Unless otherwise specified, you carry out these actions on the host on which the additional name server is to run.

Prerequisites

- You are already operating one name server. If this is not the case, you must activate a name server first (see Distributed System with Name Server [Page 26]).
- TREX is installed on the host on which the additional name server is to run.

Configuration

✓	**Action**
	Stop the TREX demon (See Starting and Stopping the TREX Service [Page 85]).
	Configure the TREX demon so that it starts the additional name server (see Configuring the TREX Demon [Page 50]).
	Register the name server with all remaining TREX servers [Page 51].
	Start the TREX demon (See Starting and Stopping the TREX Service [Page 85]).
	Activate the name server [Page 52].
	Test name server replication [Page 53].
	Only on the portal server, and only if the name server is activated on Content Management-side: • Register the additional name server with Content Management (see Registering the Name Server with Content Management [Page 55]). • Restart the servlet engine

Configuration

Purpose

The following sections describe the individual configuration steps that are necessary for name server replication.

Configuring the TREX Demon

Use

You have to modify the `TREXDaemon.in` configuration file on the host on which the additional name server is to run.

Prerequisites

The TREX demon is stopped (see Starting and Stopping the TREX Service [Page 85]).

Procedure

1. Open the `<TREX_Directory>\TREXDaemon.ini` configuration file with a text editor.
2. Create a new section for the name server. For the structure of the sections, see Configuring the TREX Demon [Page 30].
3. In the `[daemon]` section, in the `programs` parameter, add the `nameserver` entry.
4. Save the file and close the text editor.

Example

The example below depicts the section for the name server in the configuration file.

```
[daemon]
programs = nameserver, indexserver, preprocessor

[nameserver]
executable=TREXNameServer
arguments=
startdir=D:\my_trex_path\SAP\TREX_6
instances=1
...
```

Registering the Name Server with All Remaining TREX Servers

Use

You register the additional name server with all remaining TREX servers. This involves modifying the `TrexConfigMgr.ini` configuration file on all TREX hosts.

Prerequisites

The TREX demon is stopped on all hosts (see Starting and Stopping the TREX Service [Page 85]).

Procedure

1. Open the `<TREX_Directory>\TREXConfigMgr.ini` configuration file with a text editor.
2. The existing name server is entered in the `[all]` section, in the `PNS-address` line. Add the address of the additional name server.

   ```
   PNS-address = tcpip://<hostname1>:<port>,tcpip://<hostname2>:<port>
   ```

   Example

   ```
   [all]
   PNS-address = tcpip://mytrexhost1:8355,tcpip://mytrexhost2:8355
   ...
   ```

   Enter the name servers in the same order on all hosts.
3. Save the file and close the text editor.

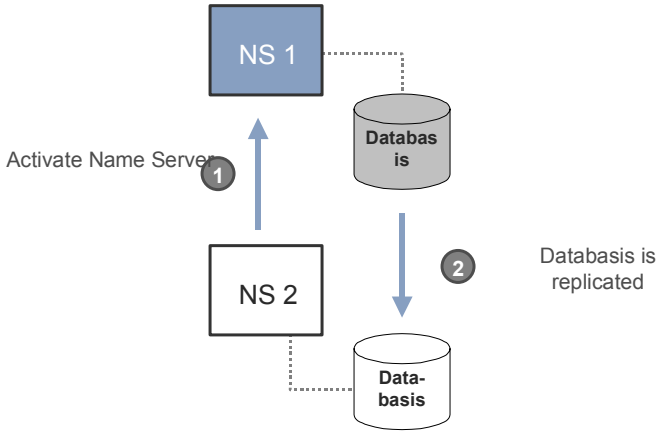
Result

Start the TREX demon on all hosts.

Activating the New Name Server

Use

You have to activate the additional name server once. Activating the new name server registers it with the existing name server Name server replication begins to take place as soon as you activate the name server. The new name server then receives all information that is written to the databasis of the existing name server during routine operation.



Prerequisites

The name server is running (that is, the TREX demon has started on the hosts involved).

Procedure

1. Start name server administration:

   `<TREX_Directory>\TREXNameServerGUI.py`

2. Choose *Change address* and enter the address of the existing name server:

   `tcpip://<hostname>:<port>`

   Example: **`tcpip://mytrexhost1:8355`**

3. Choose *OK*.

4. Choose *setActive* and enter the following data:

Field	Entry
service	**`nameserver`**
address	Address of the new name server (the one you want to register). `<hostname>:<port>`

   service: **`nameserver`**

   address: **`mytrexhost2:8355`**

5. Choose *SEND*.

 Testing Name Server Replication

Use

When you have finished the configuration steps, you can check whether replication is working correctly.

Prerequisites

The name servers are running (that is, the TREX demon has started on the hosts involved).

Procedure

1. Start name server administration.

   `<TREX_Directory>\TREXNameServerGUI.py`

2. Choose *Change address* and enter the address of the first name server.

   `tcpip://<hostname>:<port>`

   Example: **`tcpip://mytrexhost1:8355`**

3. Choose *OK*.

4. Choose *putServData* and enter the following data:

Field	Entry
service	`testservice`
address	`testaddress`
data	`[my_testindex]` `myparam=myvalue`

5. Choose *SEND.*
6. Choose *Change address* and enter the address of the second name server.

   Example: `tcpip://mytrexhost2:8355`
7. Choose *OK.*
8. Choose *getServInfo* and enter the following data:

Field	Entry
service	`testservice`
index	`my_testindex`

9. Choose *SEND.*

Result

The output area should display the following data:

```
testservice = testaddress
[my_testindex]
myparam=myvalue
```

Registering the Name Server with Content Management

Use

If the name server is activated on Content Management-side, you have to modify the configuration of Content Management. This registers all alternative name servers with Content Management.

Prerequisites

The portal and Content Management are installed and running. You can log onto the portal and have the role *KM Admin.*

Procedure

1. Log on to the portal and choose *KM Admin* → *Configuration* → *TREX* → *TREX Java Client* from the top-level navigation bar.
2. Choose *Name Server.* Edit the *nameserver* entry as follows:

Parameter	Entry
`Backup Servers`	Enter the addresses of all alternative name servers. If you are setting up name server replication for the first time, enter the following addresses: • The address of the existing name server t hat is already entered into the `Address` parameter. • The address of the additional name server. The address format is `tcpip://<hostname1>.<domain>:<port>,<hostname2>.<domain>:<port>.` Unless you have specified another port during the installation of TREX, the port number is 8355.  Enter the name servers in the same order as in the `TrexConfigMgr.ini` file.

Example

You have entered two name servers into the `TrexConfigMgr.ini` configuration file. The analogous entry in the portal is:

Parameter	Value
`Address`	**`tcpip://mytrexhost1.mydomain.com:8355`**
`Backup Servers`	**`tcpip://mytrexhost1.mydomain.com:8355,tcpip://mytrexhost2.mydomain.com:8355`**

Result

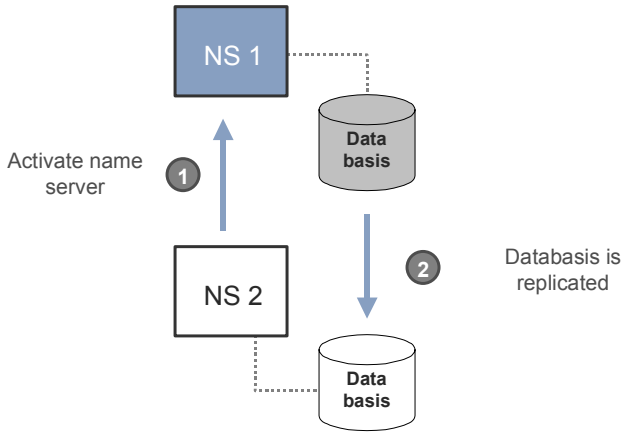
Restart the servlet engine and the portal server so that the changes take effect.

How does name server replication work?

Name server replication makes sure that all operating name servers have the same databasis status. The following sections explain the situations in which the databasis is replicated.

When a name server is activated

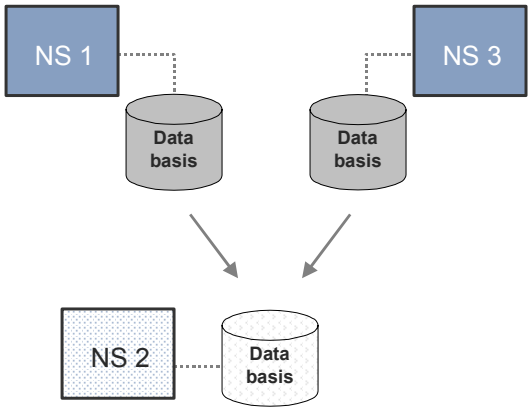
When you activate a name server, you ensure that the name servers recognize each other and that replication starts.



When a name server is started

When a name server starts, it requests the current databasis from all other name servers. The name servers mark the newly started name server as active in their databases and then send their databases to the newly started name server. Since the databases of all operating name servers contain the same information, the newly started name server only imports the first databasis that it receives.

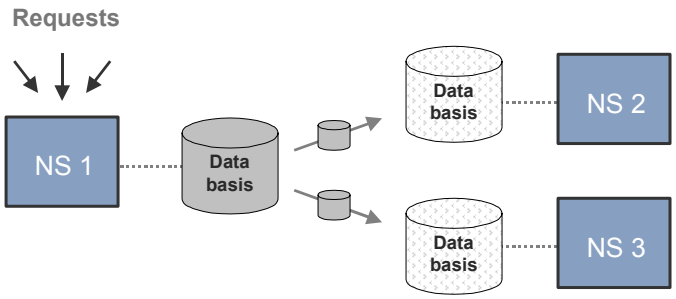
The graphic below depicts a scenario with three name servers. Name servers 1 and 3 are active and have the same data status. When name server 2 starts, it receives the current databasis from both servers.



In a further TREX release, it may be necessary for the newly started name server to merge two different databases. This is why the replication process already allows the newly started name server to receive the databases of all operating name servers.

During routine operation

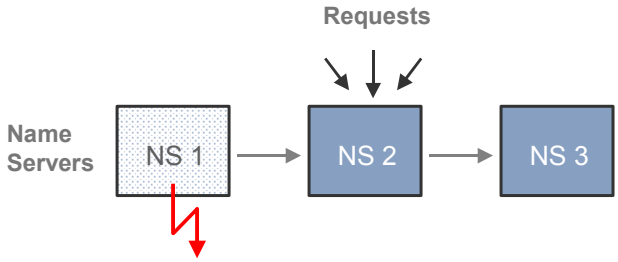
As soon as the databasis of a name server changes, it replicates the changes to the other name servers.



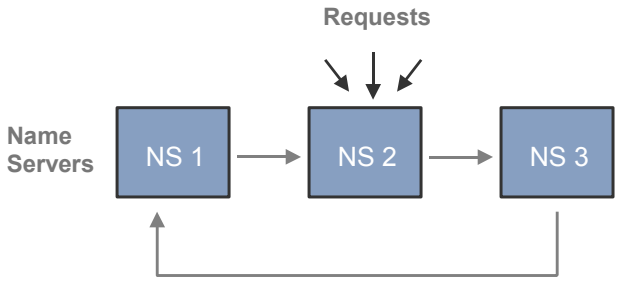
If a name server is not available, the name server that is replicating the changes receives a message reporting this fact. The name server marks the unavailable name server as inactive in its databasis. It then sends a message to the other active name servers so that they know about the change in status of the name server that has just gone down.

What happens when one of the name servers goes down?

If the name server currently responsible for communication goes down, one of the other name servers takes over this task. The TREX servers send their requests to the next name server that is entered in the configuration file. If the name server is also activated on Content Management-side, the portal sends its requests to the next name server that is entered as a backup server.

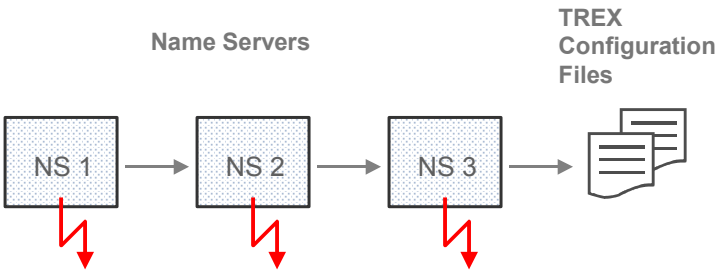


Communication then takes place using the alternative name server until it is stopped or goes down. When this happens, the next name server in the list takes over the communication role. This process continues as long as there are still entries in the name server list. When the end of the name server list is reached, the system returns to the first entry in the list.



If none of the name servers is available, no load distribution takes place. However, the TREX servers and Content Management can continue to communicate with each other.

- The TREX servers communicate using the information stored in their configuration files.
- If Content Management uses the name server, it continually updates its configuration using information received from the name server. If no name server is available, Content Management uses the information it received most recently.



The TREX servers and Content Management check regularly to see whether a name server is available again. If a name server is available, communication takes place using it.

A name server that is not available is not used for replication. When the TREX demon restarts the name server, the name server receives the current databasis and takes part in the replication process again.

Index Replication

Purpose

You can use index replication to make one index available on more than one search server. This improves search and text-mining performance, and ensures the availability of indexes for searching.

The following sections describe how you set up index replication and how index replication works. They also contain information on deleting and initializing a replicated index.

Implementing Index Replication

Purpose

TREX offers automatic and manual index replication. This document only explains how you implement automatic index replication. Use the checklist below for the implementation of index replication. Keep to the sequence specified in the table.

Prerequisites

- You are using a name server.
- TREX has been installed on the index servers involved (master and slaves).

Configuration

✓	Action
	On the name server:
	If you want original indexes to be created only on the master and not on the slaves, set the master as the default index server [Page 64].
	On the master:
	Activate the Python extension handler [Page 70].
	Activate automatic export [Page 66]. This also defines whether all indexes or only selected indexes are exported.
	Stop and restart the TREX demon (See Starting and Stopping the TREX Service [Page 85]).
	Share the replication directory [Page 68] so that all slaves can access it. The replication directory contains the backups of the indexes.
	On all slaves:
	Activate the Python extension handler [Page 70].
	Activate the slave extension [Page 71].
	Configure automatic import [Page 72]. This defines which indexes are imported and the time interval between imports.
	Stop the TREX demon (See Starting and Stopping the TREX Service [Page 85]).
	Configure the TREX demon [Page 79] so that it also starts the import demon.
	Start the TREX demon (See Starting and Stopping the TREX Service [Page 85]).

Configuration

Purpose

The following sections describe the individual configuration steps that are necessary for index replication.

Summary

Purpose

The summary contains all parameters that you configure.

Configuration

On the name server:

✓	Step	Summary
	If indexes are only to be created on the master, mark the master as the default index server.	Start `<TREX_Directory>\TREXNameServerGUI.py`. Method *setAsDefault* *service* = **`indexserver`** *address* = **`<address_of_master>`**, for example, `mytrexhost1:8351`

On the master:

✓	Step	Summary

	Activate the Python extension handler.	Edit the `<TREX_Directory>\TREXExtensions.ini` file:
		`[activate]`
		`imsapi`
		`[extensionhandlers]`
		`trexxpy`

	Activate automatic export.	Edit `<TREX_Directory>\extensions.py`
		or
		`<TREX_Index_Directory><Index>\extensions.py:`
		`# replication master extension:`
		`if 1`

	Stop and restart the TREX demon.	*Start* → *Programs* → *SAP TREX* → *TREX Service* → *stop* and *Start* → *Programs* → *SAP TREX* → *TREX Service* → *start*
	Share the replication directory.	Share `<TREX_Backup_Directory>\replication`.

On all slaves:

✓	Action	Summary

	Activate the Python extension handler.	Edit the `<TREX_Directory>\TREXExtensions.ini` file:
		`[activate]`
		`imsapi`
		`[extensionhandlers]`
		`trexxpy`

	Activate the slave extension.	Edit the `<TREX_Directory>\extensions.py` file: `# replication slave extension:` **`if 1`**
	Configure automatic import	Edit the `<TREX_Directory>\TREXImport.ini` file: `[autoimport]` **`sourcepath=<Replication_Directory>`** **`indexes=<Index1>,<Index2>,...`** **`active=yes`**
	Stop the TREX demon.	*Start → Programs → SAP TREX → TREX Service → stop*
	Configure the TREX demon	Edit the `<TREX_Directory>\TREXDaemon.ini` file: `[daemon]` `programs = indexserver, preprocessor,` **`importdaemon`** **`[importdaemon]`** **`executable=python`** **`arguments=import.py --daemon`** **`startdir=<TREX_Directory>\python_support\imp_exp_idx`** **`instances=1`**
	Start the TREX demon.	*Start → Programs → SAP TREX → TREX Service → start*

Setting the Master as the Default Index Server

Use

The name server dictates the index servers on which new indexes are created. By default, it distributes requests to create new indexes amongst all known index servers.

In a system with index replication, the slaves should normally contain copies of indexes, but no original indexes. You can configure the name server appropriately by indicating that the master server is the default index server.

The name server then only forwards create index requests to the default index server. If no index server is indicated as the default index server, all index servers are treated as default.

Prerequisites

The name server is running (that is, the TREX demon has started).

Procedure

1. Start name server administration.

   `<TREX_Directory>\TREXNameServerGUI.py`

2. Choose *Change address* and enter the address of the name server.

   Example: **`tcpip://mytrexhost1:8355`**

3. Choose *OK*.

4. Choose *setAsDefault* and enter the following data:

Field	Entry
service	**`indexserver`**
address	Address of the master in the format `<hostname>:<port>`. Example: **`mytrexhost1:8351`**

5. Choose *SEND*.

Configuration of the Master

Purpose

The following sections describe how you configure the master.

Activating the Python Extension Handler

Procedure

1. Navigate to the TREX directory.
2. Open the `TREXExtensions.ini` configuration file with a text editor.
3. In the `[activate]` section, add the line **`imsapi`** and/or remove the hash sign (#).

   ```
   [activate]
   imsapi
   ```

4. In the `[extensionhandlers]` section, add the line **`trexxpy`** and/or remove the hash sign (#).

   ```
   [extensionhandlers]
   trexxpy
   ```

5. Save the file and close the text editor.

Activating Automatic Export

Use

You can use the `extensions.py` configuration file to configure automatic export on the master. You can replicate all indexes or only selected indexes.

- If you want all indexes to be replicated, change the `extensions.py` configuration file directly in the TREX directory.
- If you only want to replicate selected indexes, copy the `extensions.py` configuration file to the relevant index directory, and change the copy.

Replicating All Indexes

1. Open the `<TREX_Directory>\extensions.py` configuration file with a text editor.
2. Change the `if 0:` entry in the section that relates to the export of indexes to **`if 1:`**. The section in question starts with the commentary `# replication master extension`.
3. Check the `ReplicationMaster(<number-of-backupsper-index>,0)` call.

   The first parameter defines the number of backups that are stored for each index. By default, the last two backups are stored. You can change this number if necessary.

```
# replication master extension:
  ...
  # ------------------------------
  if 1:
      sys.path.append(os.path.join(os.getenv('SAP_Retrieval_Path'),
'extensions','replication'))
      from master import ReplicationMaster
      trexx.registerExtension(trexx.EXTCLASS_ADMIN,
ReplicationMaster(2,0))
```

4. Save the file and close the text editor.
5. Stop and restart the TREX demon (see Starting and Stopping the TREX Service [Page 85]).

Replicating Selected Indexes

1. Copy the `<TREX_Directory>\extensions.py` configuration file to the directory of the index to be replicated:

   `<TREX_Index_Directory>\<Index_ID>\extensions.py`

   In this case, only change the copy of the configuration file. Do not change the version in the TREX directory.

   The superordinate index directory that contains the subdirectories of the individual indexes is `<TREX_Directory>\index` by default. If you do not know the name of the index directory, look it up in the `<TREX_Directory>\bartho.ini` configuration file (`[Paths]` section, `data` parameter).

2. Open the copied file using a text editor. Make the same changes as described in the *Replicating All Indexes* section.
3. Save the file and close the text editor.
4. Stop and restart the TREX demon (see Starting and Stopping the TREX Service [Page 85]).

Sharing the Replication Directory

Use

The backups of the indexes are created in the replication directory on the master. The replication directory is a subdirectory of the backup directory.

`<TREX_Backup_Directory>\replication`

All slaves need read- and write-access to the replication directory on the master. In particular, the user that starts the import demon has to have these access permissions.

Later on, you can configure the TREX demon so that it also starts the import demon. Thus, the import demon runs on the same user as the TREX demon. On Windows, this is the user that you defined for the TREX service (see Changing the User for the TREX Service [Page 42]).

By default, the backup directory is `<TREX_Directory>\backup`. If you do not know the name of the backup directory, look it up in the `<TREX_Directory>\bartho.ini` configuration file (`[Paths]` section, `backup` parameter).

Procedure

Share the `<TREX_Backup_Directory>\replication` replication directory and all subdirectories. Give read and write permissions (Full Control) to the user that starts the import demon.

Structure of the Replication Directory

The replication directory has the following structure:

```
<TREX_Backup_Directory>
    replication
        <Index_ID>_<Sequential_Number>
            <Index_ID><Language1>.zip
            <Index_ID><Language2>.zip
            ...
```

Explanation

A separate directory is created for each security copy of an index. The names of these index backup directories are structured according to the following schema:

```
<TREX_Backup_Directory>\replication\<Index_ID>_<Sequential_Number>
```

The directories are numbered sequentially. This means that you can always determine which is the most current version of the index in question.

A separate zip file is created for each language version of an index. The names of the zip files are structured according to the following schema:

```
<Index_ID><Language_Key>.zip
```

If the index backup directory contains a file with the name `finished`, the export of the index in question has been completed. The index can now be imported to the slaves.

Example

The directory structure below contains subdirectories for an index with the ID `TREX`. Three backups have already been created for this index. Two subdirectories contain complete backups. The current export has not yet been completed, since the `finished` file has not yet been created.

```
replication
    TREX_1
        TREXDE.zip
        TREXEN.zip
        finished
    TREX_2
        TREXDE.zip
        TREXEN.zip
        finished
    TREX_3
        TREXDE.zip
```

The replication directory contains as many subdirectories per index as backups stored. The number of backups can be configured in the `extensions.py` file (see Configuring Automatic Export [Page 66]).

Configuration of the Slaves

Purpose

The following sections describe how you configure the slaves.

Activating the Python Extension Handler

Procedure

1. Navigate to the TREX directory.
2. Open the `TREXExtensions.ini` configuration file with a text editor.
3. In the `[activate]` section, add the line **`imsapi`** and/or remove the hash sign (#).

   ```
   [activate]
   imsapi
   ```

4. In the `[extensionhandlers]` section, add the line **`trexxpy`** and/or remove the hash sign (#).

   ```
   [extensionhandlers]
   trexxpy
   ```

5. Save the file and close the text editor.

Activating the Slave Extension

Procedure

1. Open the `<TREX_Directory>\extensions.py` configuration file with a text editor.
2. Change the `if 0:` entry in the section that relates to the slave extension to **`if 1:`**. The section in question starts with the commentary `# replication slave extension`.

   After the Change:

```
# replication slave extension
  ...
  # ------------------------------
  if 1:
      sys.path.append(os.path.join(os.getenv('SAP_Retrieval_Path'),
'extensions','replication'))
      ...
```

3. Save the file and close the text editor.

Result

The changes take effect when the TREX demon is next started.

Configuring Automatic Import

Use

An import demon must run on all slaves. The import demon checks at regular intervals to see whether a new index version is available in the replication directory of the master. If a new version is available and the time for the next import has arrived, the demon starts the import.

You configure the import demon using the `TREXImport.ini` file. Modify the configuration files on all slaves before starting the demon. You have to define at least the replication directory and the indexes to be imported.

Prerequisites

The indexes to be imported by slaves must already have been created on the master.

Procedure

1. Open the `<TREX_Directory>\TREXImport.ini` configuration file on the slave with a text editor.
2. Make the desired changes (see TREXImport.ini Configuration File [Page 73]).
3. Save the file and close the text editor.

TREXImport.ini Configuration File

The `TREXImport.ini` configuration file consists of several sections.

- The `[global]` section defines parameters that are valid for the import demon.
- The `[autoimport]` section defines parameters that are valid for the indexes to be imported.
- The `[index_<index-ID>]` sections define parameters that are valid for an individual index that is to be imported.

Structure

```
[global]
# wake up every
sleeptime=<seconds>

[autoimport]
sourcepath=\\<hostname>\<share>
indices=<index_ID1>,<index_ID2>
importevery=<minutes>
active=<yes|no>

[index_<index_ID>]
sourcepath=\\<hostname>\<share>
importevery=<minutes>
# set by daemon
currentversion=<number>
currentversiontimestamp=<timestamp>
```

You can import indexes manually by calling a Python script as well as automatically using the import demon. The configuration file controls both types of import. The following description specifies the individual parameters, regardless of whether they are valid for manual, automatic, or both types of import.

Apart from this, this document only describes automatic import.

[global] Section

This section describes parameters for the import demon.

`sleeptime`

Automatic import.

Time in seconds that the demon waits before becoming active again. After this amount of time, the demon again checks to see whether there are new index versions in the replication directory.

If you specify no value, the default (120 seconds) is used.

[autoimport] Section

This section defines parameters for all indexes to be imported.

`sourcepath`

Manual and automatic import.

Path to the backups of the original indexes. Enter the replication directory on the master here.

Example

`\\indexserver1\program files\sap\saptrex\backup\replication`

This path is valid for all indexes by default. If an index is located in another source directory, create a separate section for the index in question, and specify the source directory there.

`indices`

Automatic import.

IDs of the indexes that are to be imported automatically. Separate the individual indexes using commas.

You have to specify all indexes to be imported here – even those for which you create a separate section.

`importevery`

Automatic import.

Time in minutes for which the import demon waits before it imports a new index version. You can use this parameter to control the time interval between two import processes. If you set this parameter to 0, the import is triggered as soon as a new index version is available.

An import causes CPU load on the slave. Depending on the size of the index, there will also be a corresponding network load when the index is copied from the master to the slave.

`active`

Automatic import.

Value: `yes` or `no`.

Specifies whether the import demon is to be active. Indexes are only imported automatically if you set this parameter to `yes`.

[index_<index_ID>] Section

This section defines parameters for an individual index that is to be imported. You can make special settings, which are different from the general settings in the `[autoimport]` section, for each index. Parameters defined in this section take priority over the parameters with the same names in the `[autoimport]` section.

`sourcepath`

Manual and automatic import.

Path to the backups of the original indexes. Enter the replication directory on the master here. Example:

```
\\indexserver1\program files\sap\saptrex\backup\replication
```

This specification has priority over the entry in the `[autoimport]` section.

`importevery`

Automatic import.

Time in minutes for which the import demon waits before it imports a new index version. You can use this parameter to control the time interval between two import processes. If you set this parameter to 0, the import is triggered as soon as a new index version is available.

This specification has priority over the entry in the `[autoimport]` section.

An import causes CPU load on the slave. Depending on the size of the index, there will also be a corresponding network load when the index is copied from the master to the slave.

`currentversion`

Automatic import.

Version number of the last index version imported.

This parameter is filled by the import demon during routine operation.

`currentversiontimestamp`

Automatic import.

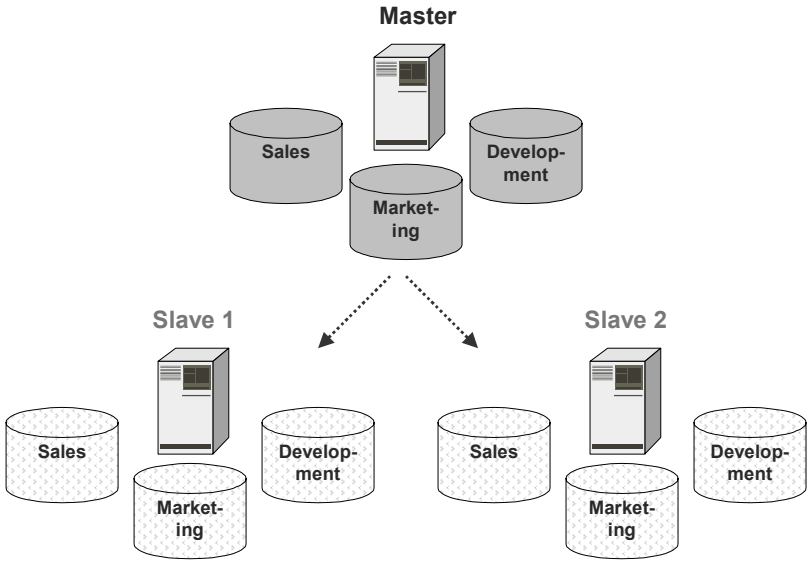
Time at which the last import process took place.

This parameter is filled by the import demon during routine operation.

Example Configuration

Example 1

The graphic below depicts a scenario with one master and two slaves. The master has three indexes that are to be imported by all slaves.



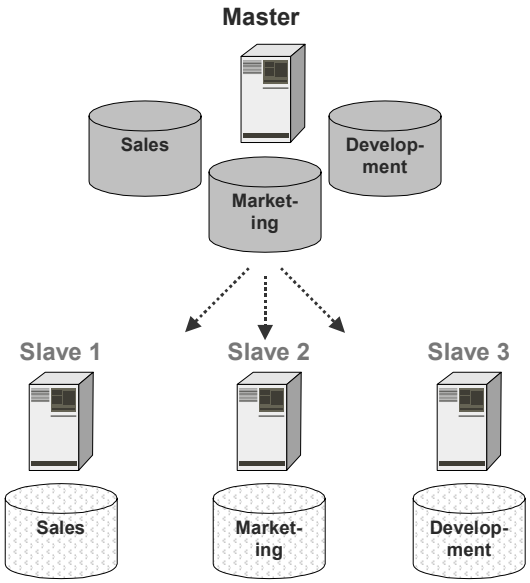
The import demon is to check for a new version every minute. The import is to be triggered as soon as a new version is available.

For this example, the `TREXImport.ini` configuration file on all slaves should be as follows:

```
[global]
# wake up every
sleeptime=60

[autoimport]
sourcepath=\\indexserver1\replication
indices=sales,marketing,development
importevery=0
active=yes
```

Example 2

The graphic below depicts a scenario with one master and three slaves. Of the three indexes that are on the master, each slave should import only one.



For this example, the `TREXImport.ini` configuration file differs from slave to slave in the `[autoimport]` section.

Slave 1

```
[autoimport]
sourcepath=\\indexserver1\replication
indices=sales
importevery=0
active=yes
...
```

Slave 2

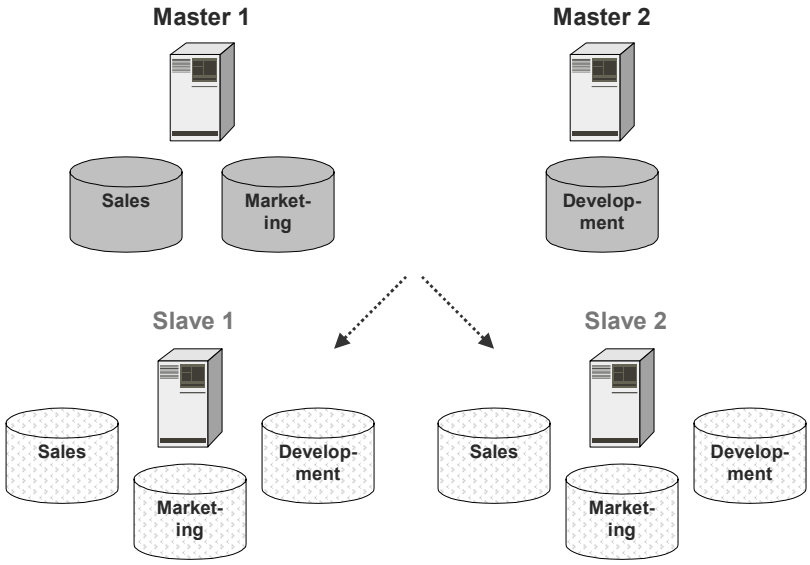
```
[autoimport]
sourcepath=\\indexserver1\replication
indices=marketing
importevery=0
active=yes
...
```

Slave 3

```
[autoimport]
sourcepath=\\indexserver1\replication
indices=development
importevery=0
active=yes
...
```

Example 3

The graphic below depicts a scenario with two masters and two slaves. Each master has indexes that are to be imported by both slaves.



For this example, the `TREXImport.ini` configuration file on both slaves could be as follows:

```
[global]
# wake up every
sleeptime=60

[autoimport]
sourcepath=\\indexserver1\replication
indices=sales,marketing,development
importevery=0
active=yes
```

```
[index_development]
sourcepath=\\indexserver2\replication
importevery=
# set by daemon
currentversion=<number>
currentversiontimestamp=Time
```

Configuring the TREX Demon

Use

You can configure the TREX demon so that it also starts the import demon as well as the TREX servers. To do this, modify the `TREXDaemon.ini` configuration file.

Prerequisites

The TREX demon is stopped on the slave (see Starting and Stopping the TREX Service [Page 85]).

Procedure

1. Open the `<TREX_Directory>\TREXDaemon.ini` configuration file with a text editor.
2. Create a new section for the import demon.

   ```
   [importdaemon]
   executable=python
   arguments=import.py --daemon
   startdir=<TREX_Directory>\python_support\imp_exp_idx
   instances=1
   ```

   For information on the meanings of the parameters, see Configuring the TREX Demon [Page 30].
3. In the `[daemon]` section, in the `programs` parameter, modify the `importdaemon` entry.
4. Save the file and close the text editor.

Result

The changes take effect when the TREX demon is next started.

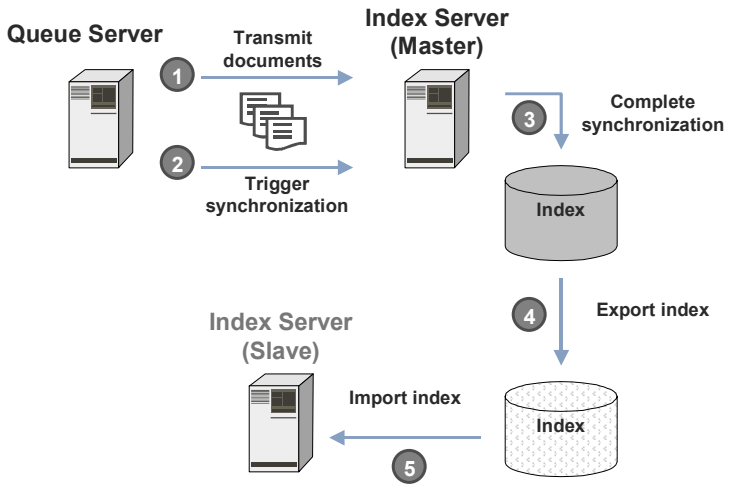
Example

The example below depicts the section for the import demon in the configuration file.

```
[daemon]
programs = indexserver, preprocessor, importdaemon

[importdaemon]
executable=python
arguments=import.py --daemon
startdir=D:\my_trex_path\SAP\TREX_6\python_support\imp_exp_idx
instances=1
...
```

How does index replication work?

The graphic below gives an overview of how index replication works during routine operation.



Export

All index changes are carried out on the server on which the original index is stored. As soon as the index change has been completed, the index is exported automatically. The security copy is written to the replication directory on the master.

When is the export triggered?

Firstly, the queue server gathers together all documents to be processed and transmits them to the index server (1). The index server then carries out the actual processing of the documents (it inserts the documents into the index, or removes them from the index). The process of insertion or removal is also called synchronization (2).

In order to optimize the performance during the indexing process, documents are transmitted to the index server in groups rather than individually, and are processed there.

You can control the following on the queue server with the help of two parameters:

- The number of documents that are transmitted to the index server in one go
- The number of transmissions after which the actual indexing/deindexing (synchronization) process is triggered

For example, the queue server can forward 1000 documents to the index server at a time. After it has transmitted three lots of 1000 documents, the queue server triggers synchronization on the index server.

The timing of the synchronization is decisive for the export. The export is triggered each time that synchronization has been completed (3 and 4).

Import

The import demon on a slave regularly checks whether a new index version is available in the replication directory. If a new version is available, and the time for the next import has been reached, the index is copied from the master to the slave, and the actual indexing process is then started (5).

Before the actual import takes place, the slave locks the index against search requests for a short amount of time. This lock is not noticed during routine operation.

When is the import triggered?

You can use the `TREXImport.ini` configuration file to control when the import of an index is triggered.

- Variant 1 – The import takes place as soon as a new index version is available on the master.
- Variant 2 – The import only takes place after a certain amount of time has passed since the last import process. In this case, the import still only takes place if a new index version is available on the master.

Synchronizing the Timing of Export and Import

Ideally, the timing of the export should be synchronized with the timing of the import. This means that a new index version is imported as soon as possible after the export, so that as current an index as possible is available on the slaves.

For performance reasons, neither the export nor the import ought to take place too frequently. We recommend that you set up the queue server so that synchronization does not take place too frequently on the master. You can control this using the queue parameters `Transmit Bulk Size`, `Synchronize Bulk Size` and `Schedule Time`. For details on the queue parameters, see the TREX administration guide.

You control the timing of the import using the parameters in the TREXImport configuration file [Page 73].

Deleting and Initializing a Replicated Index

Procedure

Before deleting or initializing a replicated index, you ought to make sure that the index servers involved (master and slaves) are running.

- If you delete an index, the index is deleted from the master and from all slaves.
- If you initialize an index, the index is initialized on the master and on all slaves.

Distributed Preprocessing

Purpose

Before TREX indexes a document, it has to preprocess it. Therefore, the queue server forwards the document to the preprocessor first. The preprocessor loads the document from the repository, filters it, and carries out a linguistic analysis. It then forwards the document to the queue server.

By default, TREX is configured so that the queue server only forwards documents to **one** preprocessor. The queue server normally uses the preprocessor that is running locally on the host.

Depending on the format and size of the documents, the preprocessing can generate as much system load as the indexing itself. If a large number of non-HTML or non-text documents are to be indexed, it may be beneficial to distribute the preprocessing amongst multiple preprocessors. This can increase data throughput for preprocessing, and thereby for the entire indexing process.

For example, you can use one or more hosts exclusively for preprocessing documents. One or more instances of the preprocessor can run on these hosts. For an example of this kind of scenario, see Error-Tolerant and Scalable System [Page 15].

If you are using multiple preprocessors for the preprocessing, you have to configure the queue server **and** the preprocessors. For information on the configuration, see SAP Note 585389.

Starting and Stopping TREX

Purpose

The following sections explain how you start and stop TREX.

Process Flow

In a distributed installation, you start TREX in the following order:

1. Firstly, on the name servers
2. Then on all other hosts

Starting and Stopping the TREX Service

Use

When TREX is installed, a TREX service that is started automatically when the host is started up, and stopped automatically when the host is shut down, is registered. As soon as this service starts, TREX is ready for operation.

You can start and stop the TREX service manually if necessary.

Starting the TREX Service

Choose *Start* → *Programs* → *SAP TREX* → *TREX Service* → S*tart*.

Stopping the TREX Service

Choose *Start* → *Programs* → *SAP TREX* → *TREX Service* → S*top*.

Certain processing steps, for example, writing an index, cannot be interrupted. Such steps are completed before the TREX service is stopped. Therefore, it can take a certain amount of time to stop the service.

With large indexes, it can take up to a few hours to stop the service if lots of documents are currently being indexed.

Starting and Stopping Individual TREX Servers

Use

The TREX demon is registered as a service when TREX is installed. The service is configured so that is automatically starts when the host is started, and automatically stops when the host is shut down. TREX is ready to use when the TREX service and Web server are running. You can start individual TREX servers for test purposes and for troubleshooting. You can then track the program output on the screen.

Starting the TREX Servers

1. Stop the TREX service (see Starting and Stopping the TREX Service [Page 85]).
2. Open a separate prompt for each TREX server.
3. Go to the TREX installation directory.
4. Select the TREX servers that you need.
   - If you have a non-distributed scenario, start the index server, preprocessor, and queue server.
   - If you have a distributed installation, start the servers that are to run on the host in question.

TREX Server	Command
Index server	`TREXIndexServer.exe`
ISAPI register (only in a distributed installation, and only on the Web server)	`TREXISAPIRegister.exe`
Name server (only in a distributed installation)	`TREXNameServer.exe`
Preprocessor	`TREXPreprocessor.exe`
Queue server	`TREXQueueServer.exe`

In the properties of the prompt, deactivate the *QuickEdit Mode* option.

Leave the prompt open. If you want, you can minimize the window so that it is shown as a pushbutton in the Windows task bar.

Stopping the TREX Servers

1. Display the window in which you started the TREX server.
2. Use CTRL+C or close the window.

Certain processing steps, for example, writing an index, cannot be interrupted. Such steps are completed before the TREX servers are stopped. Therefore, it can take a certain amount of time to stop the demon.

With large indexes, it can take up to a few hours to stop the demon if lots of documents are currently being indexed.

Do not use the Task Manager to stop the TREX server. Doing so can lead to the loss of data. Affected indexes can be irreparably damaged.

Starting and Stopping the TREX Demon in Debug Mode

Use

The TREX demon is the program that is registered as the service. If you start the TREX demon in debug mode, you can track the program output on the screen. You do this for test purposes and troubleshooting.

Starting the TREX Demon

1. Stop the TREX service (see Starting and Stopping the TREX Service [Page 85]).
2. Open a prompt and go to the TREX installation directory.
3. Execute the following command:

   ```
   TREXDaemon.exe -d
   ```

Stopping the TREX Demon

1. Display the window in which you started the TREX demon.
2. Use CTRL+C or close the window.

Certain processing steps, for example, writing an index, cannot be interrupted. Such steps are completed before the TREX demon is stopped. Therefore, it can take a certain amount of time to stop the demon.

With large indexes, it can take up to a few hours to stop the demon if a large number of documents are currently being indexed.

Do not use the Task Manager to stop the TREX demon. Doing so can lead to the loss of data. Affected indexes can be irreparably damaged.

Starting and Stopping the Web Server

Use

If necessary, you can start, restart, and stop the Web server (Microsoft Internet Information Server) manually.

Starting the Web Server

1. Choose *Start* → *Settings* → *Control Panel*. Then choose *Administrative Tools* → *Services*.
2. Use the secondary mouse button to click on *IIS Admin Service*.
3. Choose *Start* from the context menu.

If the Web server doesn't run even though you have started the service, start the Web server using a prompt.

1. Open a prompt by choosing *Start* → *Programs* → *Accessories* → *Command Prompt*.
2. Execute the following command:

```
net start w3svc
```

Restarting the Web Server

If problems occur during routine operation, you may need to restart the Web server.

1. Choose *Start* → *Programs* → *Administrative Tools* → *Internet Service Manager*.
2. Choose *Action* → *Restart IIS*.

Stopping the Web Server

1. Choose *Start* → *Settings* → *Control Panel*. Then choose *Administrative Tools* → *Services*.
2. Use the secondary mouse button to click on *IIS Admin Service*.
3. Choose *Stop* from the context menu.

Troubleshooting

Purpose

The following sections contain hints for troubleshooting.

A replicated index could not be deleted from all slaves

Use

You deleted an index that you had replicated on multiple index servers. When the deletion took place, one of the slaves was not started. Therefore, the index could not be completely deleted. There is still an index copy on the slave, and this index copy is still being recognized by the name server.

Solution: You have to delete the index copy from the slave and then remove the entry from the name server databasis.

Deleting the Index Copy from the Slave

1. On the slave, navigate to the directory `<TREX_Directory>\python_support\test_tools\lib`.
2. Start the Python script below:

   `deleteIndex.py <index_ID>`

   Example: `deleteIndex.py marketing`

   Do **not** start the script with `--cmethod=t`.

Deleting the Entry from the Name Server Databasis

1. Start name server administration.

   `<TREX_Directory>\TREXNameServerGUI.py`

2. Choose *Change address* and enter the address of the name server.

   `tcpip://<hostname>:<port>`

   Example: **`tcpip://mytrexhost1:8355`**

3. Choose *OK*.

4. Choose *delEntryData* and enter the following data:

Field	Entry
service	**`indexserver`**
address	Address of the slave from which you have deleted the index copy. **`<hostname>:<port>`**
entry	**`index_<index-ID>`**

service: **`indexserver`**

address: **`mytrexhost2:8351`**

entry: **`index_marketing`**

5. Choose *SEND*.

Frequently Asked Questions

The following sections contain the answers to questions about scaling.

General

What can I do if my system is inconsistent or has gone down (power failure)?

Start TREX in the following order:

1. Firstly, on all name servers
2. Then on all other hosts

When they start, the TREX servers report to the name server and publish their data. After the servers have started, the name server databasis should contain all current information again.

Index Replication

- How do I add an additional slave?
- How do I remove an index from the group of replicated indexes?
- How do I remove a slave from the group of servers temporarily?
- How do I remove a slave from the group of servers permanently?

How do I add an additional slave?

To add an additional index server with index replication, proceed as follows:

1. Install TREX (See Adding Hosts to a Distributed System [Page 29]).
2. Register the name server in the `TREXConfigMgr.ini` configuration file (see Registering the Name Server with Other TREX Servers [Page 38]).
3. Carry out all steps necessary for index replication on a slave (see Implementing Index Replication [Page 61]).
4. Start the TREX demon on the new host (see Starting and Stopping the TREX Service [Page 85]).
5. Start name server administration (`<TREX_Directory>\TREXNameServerGUI.py`). Check whether the new index server is registered with the name server and marked as active.

   Use the `listGroupEntries` method and enter `indexserver` as the service. Check that the new index server appears in the list and has the parameter `isActive=true`.

How do I remove an index from the group of replicated indexes?

If an index copy on a slave is no longer needed, proceed as follows:

1. Delete the index copy from the slave (see A replicated index was not deleted from all slaves [Page 90], section *Deleting the Index Copy from the Slave*).
2. Delete the index entry from the name server databasis (see A replicated index was not deleted from all slaves [Page 90], section *Deleting the Entry from the Name Server Databasis*).
3. On the slave, delete the index entry from the `<TREX_Directory>\TREXImport.ini` configuration file (`[autoimport]` section, `indices` parameter).

How do I remove a slave from the group of servers temporarily?

If you are using index replication, you may need to remove a slave from the group of servers temporarily, for example, for maintenance. If this is the case, you simply stop TREX on the slave in question (see Stopping and Starting the TREX Service [Page 85]).

The index server is then marked as inactive in the name server databasis (`isActive=false`), and is no longer contacted by the name server.

When you restart TREX on the slave, the index server reports to the name server and is marked as active again.

How do I remove a slave from the group of servers permanently?

If you are using index replication, you may want to remove a slave from the group of servers permanently. This may be because you have implemented the slave in question for test purposes only, and now want to use it in another TREX server landscape. Therefore, you no longer want the slave to import the indexes that it has been importing up to now.

Carry out the following steps on the slave:

1. Stop TREX (See Starting and Stopping the TREX Service [Page 85]).
2. Delete the name server entry (`[all]` section, `PNS-address` parameter) from the `<TREX_Directory>\TREXConfigMgr.ini` configuration file.
3. Change the `<TREX_Directory>\TREXDaemon.ini` configuration file as follows:
   - Delete the section for the import demon.
   - In the `[daemon]` section, in the `programs` parameter, delete the `importdaemon` entry.
4. Change the `<TREX_Directory>\TREXImport.ini` configuration file as follows:
   - In the `[autoimport]` section, in the `indices` parameter, delete the indexes entered there.
   - Now delete all `[index_<index-ID>]` index sections.

5. Deactivate the slave extension in the `<TREX_Directory>\extensions.py` configuration file.

   After the Change:

```
# replication slave extension
  ...
  # ------------------------------
  if 0:
      sys.path.append(os.path.join(os.getenv('SAP_Retrieval_Path'),
'extensions','replication'))
```

   The name server databasis still contains information on the index server that you have removed. However, since the index server is now marked as inactive, it is no longer contacted. Therefore, the entries in the databasis have no effect during routine operation. A further TREX release provides a script that you can use to clean up the name server databasis.

Name Server

- When do I have to use a name server?
- When do I need to replicate the name server?
- Does the name server need to run on a separate host?
- Can multiple instances of the name server run on the same host?

When do I have to use a name server?

You need a name server in the following cases:

Content Management-Side

You want to implement multiple Web servers amongst which Content Management is to distribute the requests. It is only beneficial to have multiple Web servers if you want to increase reliability for TREX and expect a large number of parallel search requests.

TREX-Side

The name server does not improve the indexing process. However, if you want to replicate indexes on multiple servers, the name server is definitely required. You can only distribute search requests amongst individual servers by using a name server.

You need index replication in the following cases:

- You are expecting a large number of search- and text-mining requests (‘see also’, classification, and so on).
- In addition to a large number of parallel requests, you are expecting frequent updates (that is, indexing will take place frequently). The CPU of the index server rises considerably when an index is updated. If this is the case, we recommend replicating the index so that the search performance is not affected.

When do I need to replicate the name server?

You should replicate the name server in all scenarios where a name server is necessary. In particular, replicate the name server if you are using index replication. This prevents a situation where search requests are no longer distributed because the sole name server is down.

Does the name server need to run on a separate host?

Not unless you want a very high degree of reliability.

Can multiple instances of the name server run on the same host?

No. You can only have one instance of a name server per host.

ISAPI Register

- What is the ISAPI register?
- When do you need this component?

What is the ISAPI register?

On Windows, the ISAPI register makes sure that the Web server reports to the name server after starting. The name server than recognizes the Web server and can forward its address.

When do you need this component?

You only need the ISAPI register if you are running on Windows **and** using a name server. If this is the case, the ISAPI register needs to run on every Web server.