Scaling Retrieval and Classification (TREX)

Release TREX 6.0 SP1
Document Version 1

Copyright

Icons in Body Text

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help* → *General Information Classes and Information Classes for Business Information Warehouse* on the first page of the any version of *SAP Library*.

Typographic Conventions

Type Style	Description
Example text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
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. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
EXAMPLE TEXT	Keys on the keyboard, for example, F2 or ENTER.

Contents

Scaling Retrieval and Classification (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 among 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 *Distributed Preprocessing for Indexing* section explains how you can distribute the preprocessing of documents among multiple hosts.
- 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.
- The appendix describes the name server administration tool.

Further Information

You also need to read the following documents before implementing a distributed system:

- For EP 6.0, the master guide *mySAP Enterprise Portal – Using SAP Enterprise Portal 6.0*
- The TREX installation guide *Installing Retrieval and Classification (TREX)* in the *SAP Enterprise Portal Installation Guide*

  This guide is located at **`http://service.sap.com/ep60`** → *Documentation & More* → *Installation* (EP 6.0) and **`http://service.sap.com/epinstall`** → *Installation, Upgrades & Patches* (EP 5.0).

The master guide contains information on implementing demo, test, and productive systems The TREX installation guide contains detailed information on planning, preparing for, and carrying out the installation of TREX.

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 other guides wherever necessary.

Overview

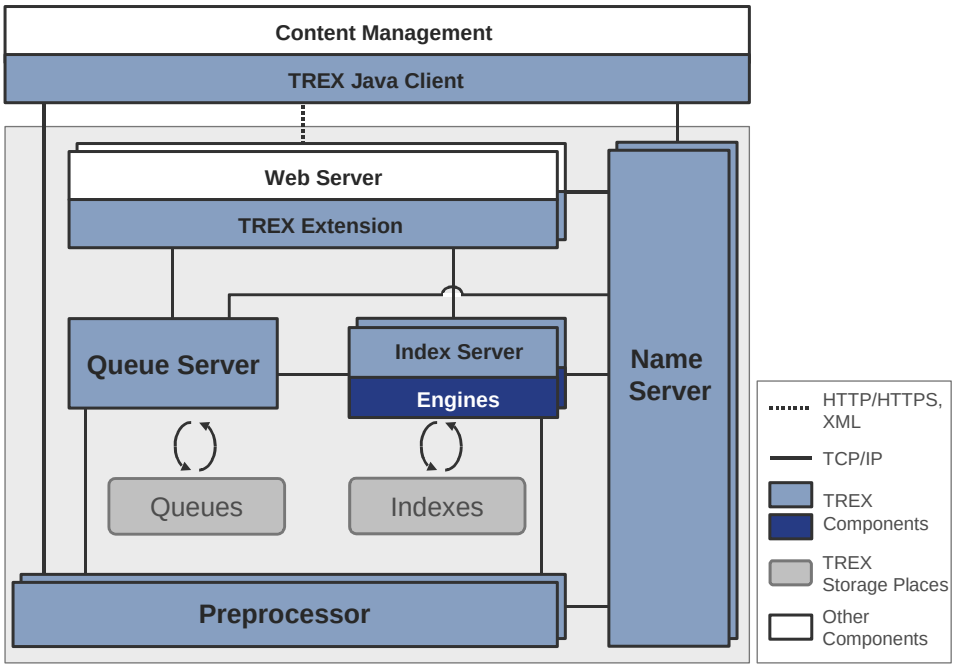
The following sections provide an overview of the central TREX components and introduce the TREX scaling concepts.

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 realized as follows:

- On Windows, as an ISAPI server extension for the Microsoft Internet Information Server
- On UNIX, as a shared library for the Apache Web 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 requests, the preprocessor carries out a linguistic analysis of search queries. The preprocessor passes the results of the analysis to the index server, which then processes the query further.
- 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 normally transferred directly to TREX. Instead, they are forwarded in the form of a URI that references the storage location of the document in question. The preprocessor resolves the URI and collects the actual document from the repository.
    - Filtering the document

      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 and converts it to Unicode format UTF 8 for further processing.
    - Analyzing the document linguistically

      The preprocessor uses a lexicon that analyzes texts in various languages.

Index Server

The index server is responsible for indexing, classifying, and searching. It receives requests and forwards them 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 (Windows Only)

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

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 Daemon

After the TREX installation, a TREX daemon runs on each host. The daemon 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 daemon.

In the standard configuration, the daemon starts all servers that have to run for a minimal TREX system. For a distributed installation, you can use the daemon’s configuration file to control which servers the daemon 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 type 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 queries will there be?
- What level of availability do you require for TREX?

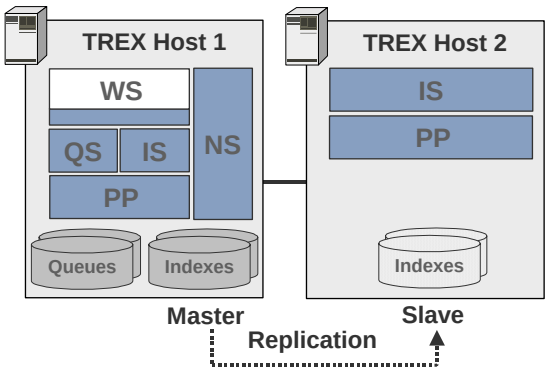
The answers to these questions determine the number of TREX hosts that you require and how the tasks are distributed among the hosts. Note the following:

- All TREX hosts must run on the same operating system platform. Mixed installations (for example, one TREX host on HP-UX and another on Windows) are not permitted
- All TREX hosts must have the same TREX release.

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

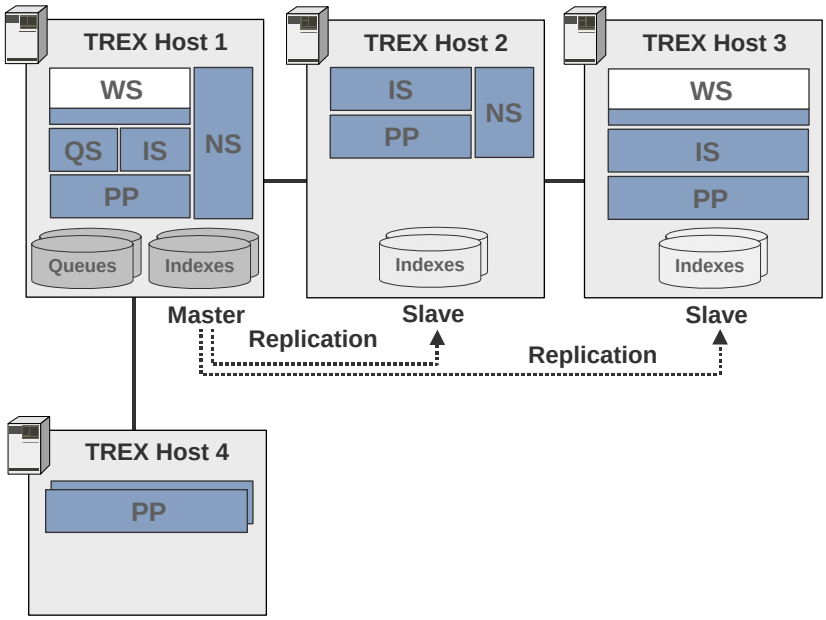
Host 2 is exclusively a search server, that is, it is only responsible for answering search requests. No indexing or classification takes place on host 2.

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.

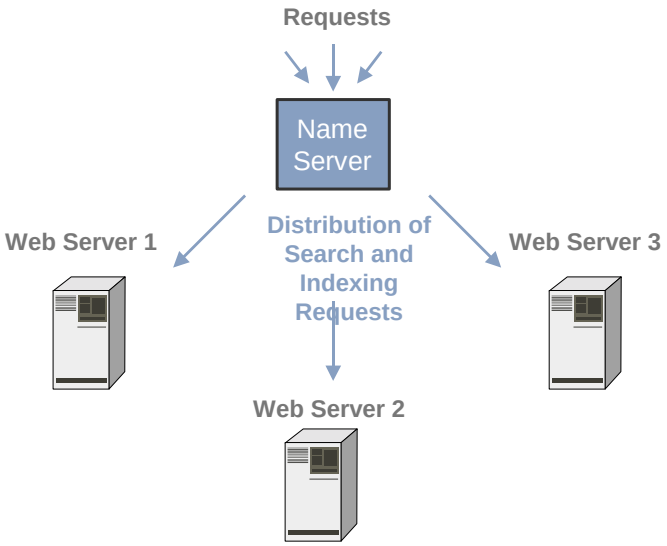
Using a name server is the only way to build up a large distributed server landscape that distributes the load amongst several servers. A name server is the prerequisite for realizing the following:

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

Load Distribution

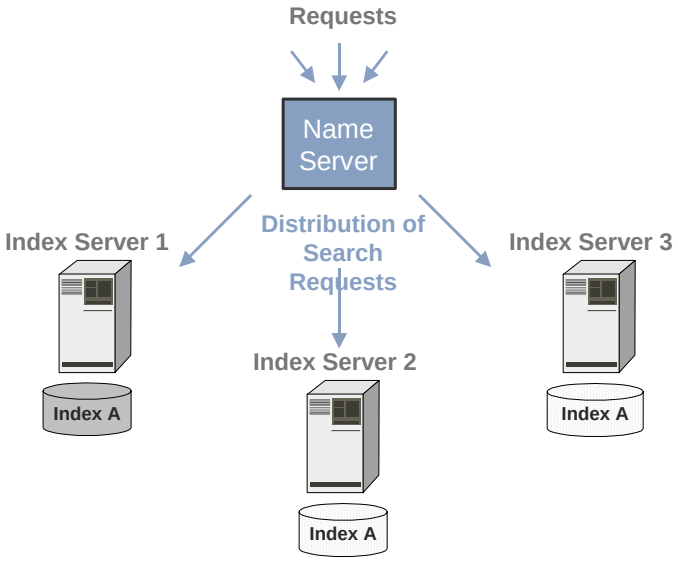
Distributing search and indexing requests amongst Web servers

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 among the Web servers being used.



Distributing Search Queries Among 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 among 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.



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

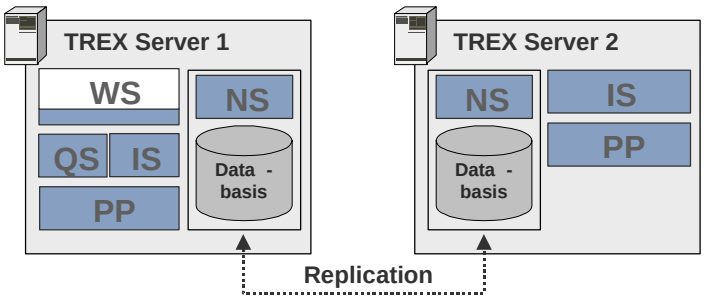
The name server offers a watchdog function that monitors all active TREX servers. The watchdog regularly checks whether the active TREX servers can be reached. If a server does not respond, the name server marks it in its databasis as inactive. This prevents the name server from forwarding the address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

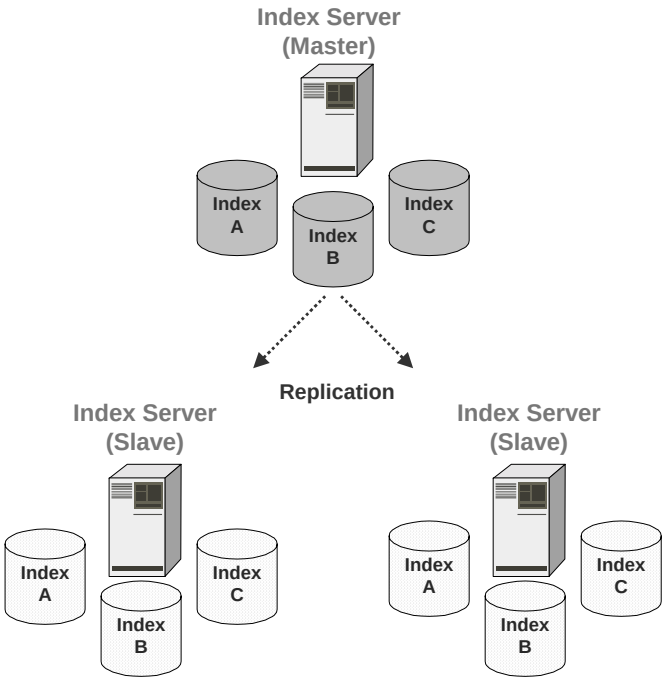
Index Replication

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 security copies of the original index runs on the master. The export takes place as soon as changes to the original index have been completed.
- An import daemon 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 daemon imports the new version.

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

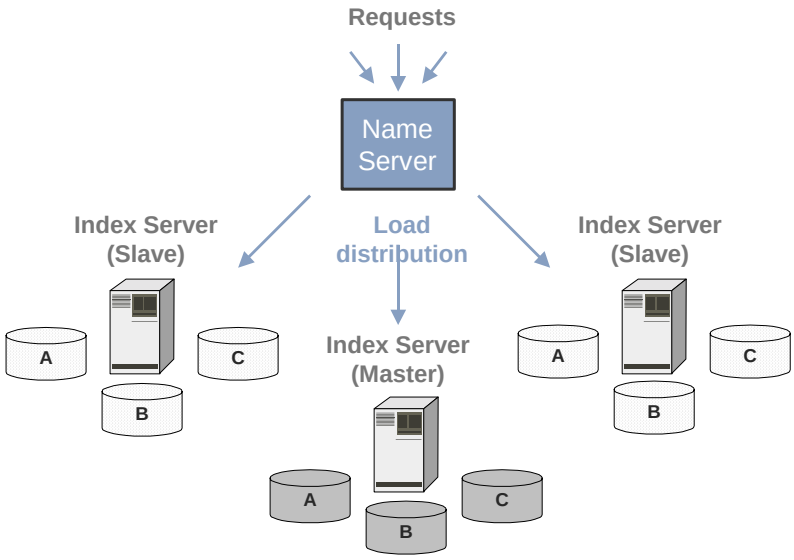
Special Considerations with Query-Based Taxonomies

If you use query-based taxonomies, a QBC replication script needs to run on the master and slaves.

This script enhances the functions of the standard replication procedure. It replicates the parts of query-based taxonomies that are not replicated by the standard procedure.

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 17]

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. These phases are always the same regardless of whether you are implementing a distributed or non-distributed system. The only difference is that there are additional configuration steps after the installation of a distributed system.

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 necessary for distribution.

For detailed information on installation, 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:

1. Print out the table below.
2. 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.

Installation

Install TREX on the involved hosts. Carry out all steps described in the TREX installation guide for the phases planning, preprocessing, installation, and post installation.

Configuration

Carry out the following configuration steps after the installation process.

✔	Action
	On all TREX hosts
	If the TREX daemon is running, stop it (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).
	Configure the TREX daemon so that it only starts the required TREX servers (see Configuring the TREX Daemon [Page 29]).
	Register the name server with all remaining TREX servers [Page 42]
	Start the TREX daemon on the individual hosts in the following order: • First on the name server • Then on all other hosts See UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).
	On the name server:
	Activate the watchdog [Page 43]
	On the portal host
	Only if you want are using 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. You must set the master as 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 59].

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 after you have activated the name server.

Prerequisites

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

Configuration

✔	**Action**
	On the TREX host
	Stop the TREX daemon (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).
	Configure the TREX daemon so that it also starts the following servers: • Name Server • ISAPI Register (Windows Only) (See Configuring the TREX Demon [Page 29]).
	Register the name server with all remaining TREX servers [Page 42].
	Stop the TREX daemon (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).
	Activate the watchdog [Page 43].
	On the portal host
	Only if you want to use 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.

Installation

Install TREX on the new host. Carry out all steps described in the TREX installation guide for the phases planning, preprocessing, installation, and post installation.

Configuration

Carry out the following configuration steps after the installation process.

✔	**Action**
	On the new TREX host
	If the TREX daemon is running, stop it (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).
	Configure the TREX daemon so that it only starts the required TREX servers (see Configuring the TREX Daemon [Page 29]).
	Register the name server with all remaining TREX servers [Page 42].
	Stop the TREX daemon (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).
	On the portal host
	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

Configuration

Purpose

The following sections contain detailed information on the individual configuration steps that may be necessary after the installation. Use the checklists [Page 25] to check which configuration steps need to be made.

Configuring the TREX Daemon

Use

The TREX daemon 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 daemon. In a distributed installation, you modify the configuration file on each host so that the daemon only starts the servers that are supposed to run on the host in question.

Prerequisites

The TREX daemon has been stopped (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).

Procedure

1. Edit the configuration file `<TREX_Directory>\TREXDaemon.ini`.
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 by the daemon. The program names of the TREX servers are: **On UNIX** – `TREXIndexServer.x`, `TREXNameServer.x`, `TREXPreprocessor.x`, `TREXQueueServer.x` **On Windows** – `TREXIndexServer`, `TREXISAPIRegister`, `TREXNameServer`, `TREXPreprocessor`, `TREXQueueServer`
`arguments=<arguments>`	Arguments fort he program call. • Enter `-i` in `TREXISAPIRegister`. • If several preprocessor instances are to run on the same host, enter the post of the relevant instance. • As a rule, no arguments are needed for the other TREX servers.
`startdir=<program directory`	Directory from which the program is called. As a rule, enter the TREX installation directory here.

Parameter	Description
`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.
`priority=<value>`	Priority of the program. Possible values: `lowest`, `low`, `normal`, `high`, `highest` You usually need to use the priority `normal`.

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

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

See also:

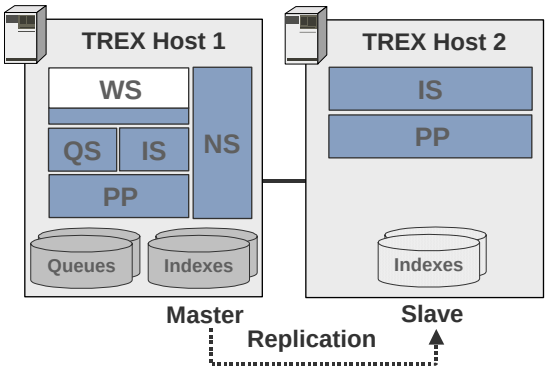
Example Configuration for a Small Distributed System [Page 31]

Example Configuration for a Large Distributed System [Page 35]

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 sections below show the relevant sections of the `TREXDaemon.ini` configuration file. The changes to the standard configuration are in bold type.

On UNIX

TREXDaemon.ini on host 1 (extract)

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


[nameserver]
executable=TREXNameServer.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal
```

```
[indexserver]
executable=TREXIndexServer.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal

[queueserver]
executable=TREXQueueServer.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal

[preprocessor]
executable=TREXPreprocessor.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal

[apache]
executable=/my_path/my_TREX_directory/Apache/bin/httpd
arguments=-d /my_path/my_TREX_directory/Apache –R
/my_path/my_TREX_directory/Apache/libexec -F
startdir=/my_path/my_TREX_directory/Apache/bin
instances=1
priority=normal
```

TREXDaemon.ini on host 2 (extract)

```
[daemon]
programs = indexserver, preprocessor

[indexserver]
executable=TREXIndexServer.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal
```

```
[preprocessor]
executable=TREXPreprocessor.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal
```

On Windows

TREXDaemon.ini on host 1 (extract)

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

[nameserver]
executable=TREXNameServer
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal

[isapiregister]
executable=TREXISAPIRegister
arguments=-i
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal

[indexserver]
executable=TREXIndexServer
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal

[queueserver]
executable=TREXQueueServer
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal
```

```
[preprocessor]
executable=TREXPreprocessor
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal
```

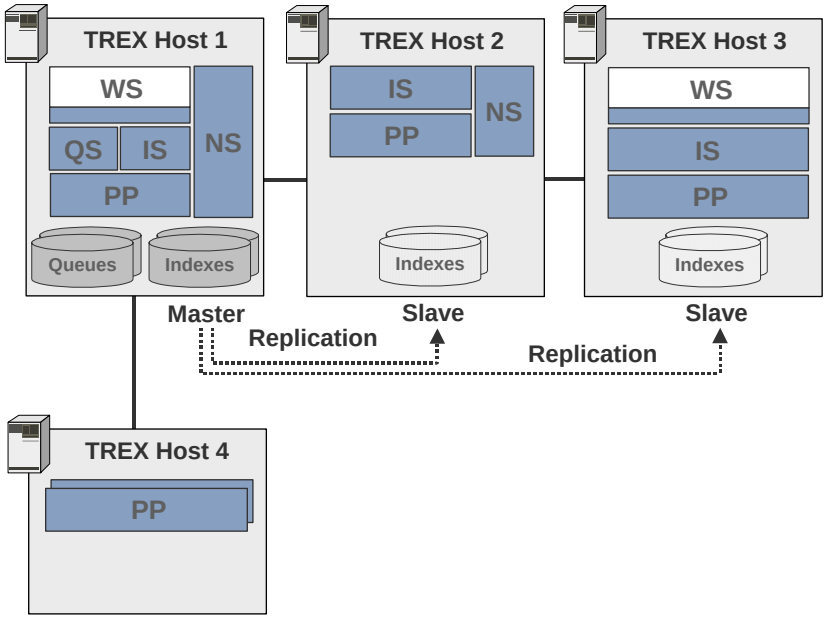
TREXDaemon.ini on host 2 (extract)

```
[daemon]
programs = indexserver, preprocessor

[indexserver]
executable=TREXIndexServer
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal

[preprocessor]
executable=TREXPreprocessor
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal
```

Example Configuration for a Large Distributed System

The graphic below displays a large distributed scenario. For an explanation of this example, 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 sections below show the relevant sections of the `TREXDaemon.ini` configuration file. The changes to the standard configuration are in bold type.

On UNIX

TREXDaemon.ini on host 1 (extract)

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


[nameserver]
executable=TREXNameServer.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal
```

```
[indexserver]
executable=TREXIndexServer.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal

[queueserver]
executable=TREXQueueServer.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal

[preprocessor]
executable=TREXPreprocessor.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal

[apache]
executable=/my_path/my_TREX_directory/Apache/bin/httpd
arguments=-d /my_path/my_TREX_directory/Apache –R
/my_path/my_TREX_directory/Apache/libexec -F
startdir=/my_path/my_TREX_directory/Apache/bin
instances=1
priority=normal
```

TREXDaemon.ini on host 2 (extract)

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

[nameserver]
executable=TREXNameServer.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal
```

```
[indexserver]
executable=TREXIndexServer.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal

[preprocessor]
executable=TREXPreprocessor.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal
```

TREXDaemon.ini on host 3 (extract)

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

[indexserver]
executable=TREXIndexServer.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal

[preprocessor]
executable=TREXPreprocessor.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal

[apache]
executable=/my_path/my_TREX_directory/Apache/bin/httpd
arguments=-d /my_path/my_TREX_directory/Apache –R
/my_path/my_TREX_directory/Apache/libexec -F
startdir=/my_path/my_TREX_directory/Apache/bin
instances=1
priority=normal
```

TREXDaemon.ini on host 4 (extract)

```
[daemon]
programs = preprocessor1, preprocessor2

[preprocessor1]
executable=TREXPreprocessor.x
arguments=-port 8357
startdir=/my_path/my_TREX_directory
instances=1
priority=normal

[preprocessor2]
executable=TREXPreprocessor.x
arguments=-port 8359
startdir=/my_path/my_TREX_directory
instances=1
priority=normal
```

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.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal
```

On Windows

TREXDaemon.ini on host 1 (extract)

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

[nameserver]
executable=TREXNameServer
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal

[isapiregister]
executable=TREXISAPIRegister
arguments=-i
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal

[indexserver]
executable=TREXIndexServer
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal

[queueserver]
executable=TREXQueueServer
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal

[preprocessor]
executable=TREXPreprocessor
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal
```

TREXDaemon.ini on host 2 (extract)

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

[nameserver]
executable=TREXNameServer
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal

[indexserver]
executable=TREXIndexServer
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal

[preprocessor]
executable=TREXPreprocessor
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal
```

TREXDaemon.ini on host 3 (extract)

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

[isapiregister]
executable=TREXISAPIRegister
arguments=-i
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal

[indexserver]
executable=TREXIndexServer
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal
```

```
[preprocessor]
executable=TREXPreprocessor
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal
```

TREXDaemon.ini on host 4 (extract)

```
[daemon]
programs = preprocessor1, preprocessor2

[preprocessor1]
executable=TREXPreprocessor
arguments=-port 8357
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal

[preprocessor2]
executable=TREXPreprocessor
arguments=-port 8359
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal
```

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_path\my_TREX_directory
instances=1
priority=normal
```

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 defines 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 daemon has been stopped (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).

Procedure

1. Edit the configuration file `<TREX_Directory>\TREXConfigMgr.ini`.
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
   ```

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
- Preprocessor
- 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 daemon has started).

Procedure

1. Start name server administration (see Starting and Closing Name Server Administration [Page 112]).
2. Choose *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 minutes. 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 daemon has started).

Procedure

1. Start name server administration (see Starting and Closing Name Server Administration [Page 112]).
2. Choose *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	Input
Server Group	**`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. Choose *watchDog.*
6. Enter the value **`start`** in the *Action* field.
7. Choose *Send* again.

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 of system administrator.

Procedure

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

Parameter	Input
`Name Server`	Enter the address of the name server in the following format: `tcpip://<hostname>.<domain>:<port>` Unless you specified another port during the installation of TREX, the port number is 8355. Example: **`tcpip://mytrexhost1.mydomain.com:8355`**
`Backup Server`	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.
`Refresh 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 host 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.

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 24]).
- TREX is installed on the host on which the additional name server is to run.

Configuration

✔	**Action**
	On the host on which the additional name server is to run
	Stop the TREX daemon (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).
	Configure the TREX daemon so that it starts the additional name server (see Configuring the TREX Daemon [Page 50]).
	Register the name server with all remaining TREX servers [Page 51].
	Stop the TREX daemon (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).
	Activate the name server [Page 52]
	Test name server replication [Page 53].
	On the portal host
	Only if the name server is activated at Content Management level. • Register the additional name server with Content Management (see Registering the Name Server with Content Management [Page 54]). • Restart the servlet engine

Configuration

Purpose

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

Configuring the TREX Daemon

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 daemon has been stopped (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).

Procedure

1. Edit the configuration file `<TREX_Directory>\TREXDaemon.ini`.
2. Create a new section for the name server. For the structure of the sections, see Configuring the TREX Demon [Page 29].
3. In the `[daemon]` section, in the `programs` parameter, modify the `nameserver` entry.

Example

Name server section from the configuration file `TREXDaemon.ini`:

On UNIX

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


[nameserver]
executable=TREXNameServer.x
arguments=
startdir=/my_path/my_TREX_directory
instances=1
priority=normal
```

On Windows

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

[nameserver]
executable=TREXNameServer
arguments=
startdir=D:\my_path\my_TREX_directory
instances=1
priority=normal
```

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 daemon has been stopped on all hosts (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).

Procedure

1. Edit the configuration file `<TREX_Directory>\TREXConfigMgr.ini`.
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 server in the same order on all hosts.

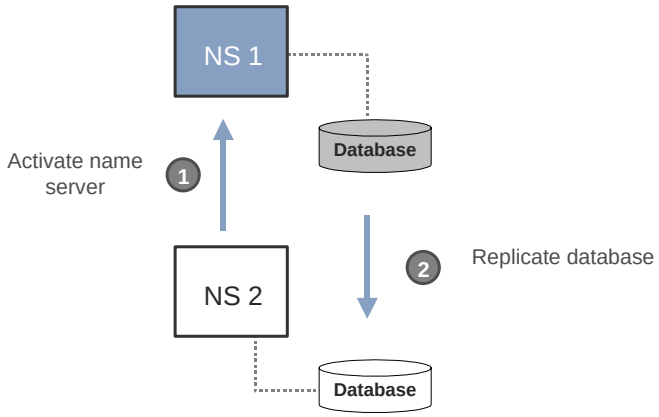
Result

Start the TREX daemon 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 daemon has started on the hosts involved).

Procedure

1. Start name server administration (see Starting and Closing Name Server Administration [Page 112]).
2. Choose *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	Input
Server Group	**`nameserver`**
Address	Address of the new name server (the one you want to register). `<hostname>:<port>`

   Server Group: **`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 server is running (that is, the TREX daemon has started on the hosts involved).

Procedure

1. Start name server administration (see Starting and Closing Name Server Administration [Page 112]).
2. Choose *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	Input
Server Group	**`testservice`**
Address	**`testaddress`**
Data	**`[my_testindex]`** **`myparam=myvalue`**

5. Choose *Send.*
6. Choose *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	Input
Server Group	**`testservice`**
Entry	**`my_testindex`**

9. Choose *Send.*

Result

The output area should display the following data:

```
OK – testaddress
```

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. You register 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 of system administrator.

Procedure

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

Parameter	Input
`Backup Server`	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. This is specified in the paramter `Name Server.` • The address of the additional name server. The address format is `tcpip://<hostname1>.<domain>:<port>,<hostname2>.<domain>:<port>.` Unless you 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
`Name Server`	**`tcpip://mytrexhost1.mydomain.com:8355`**
`Backup Server`	**`tcpip://mytrexhost1.mydomain.com:8355,tcpip://mytrexhost2.mydomain.com:8355`**

Result

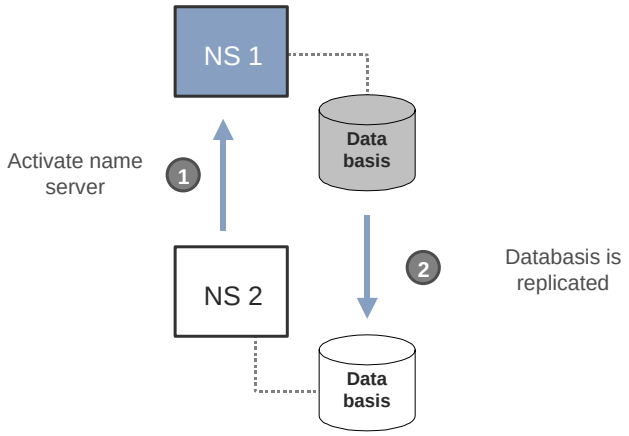
Restart the servlet engine and the portal host so that the changes take effect.

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

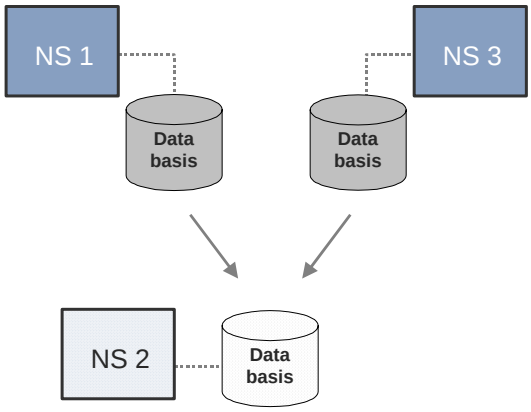
When you activate a name server, you ensure that the name servers recognize each other and that replication starts.



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.

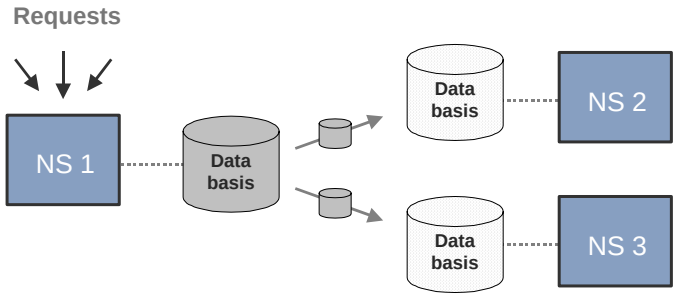
The graphic below depicts a scenario with three name servers. Name servers 1 and 3 are active and have the same data status. When name server 2 starts, it receives the current databasis from both servers.



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

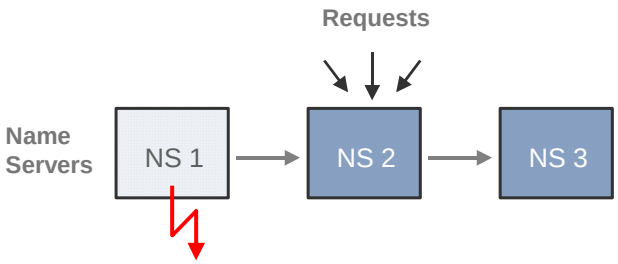
As soon as the databasis of a name server changes, it replicates the changes to the other name servers.



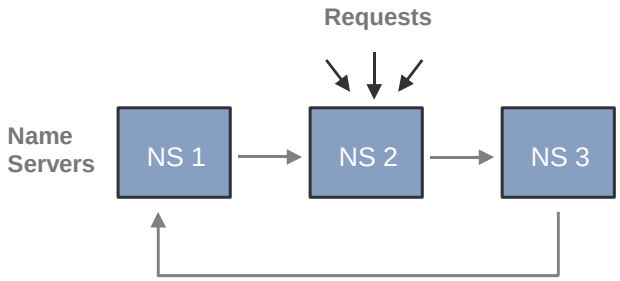
If a name server is not available, the name server that is replicating the changes receives a message reporting this fact. The name server marks the unavailable name server as inactive in its databasis. It then sends a message to the other active name servers so that they know about the change in status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

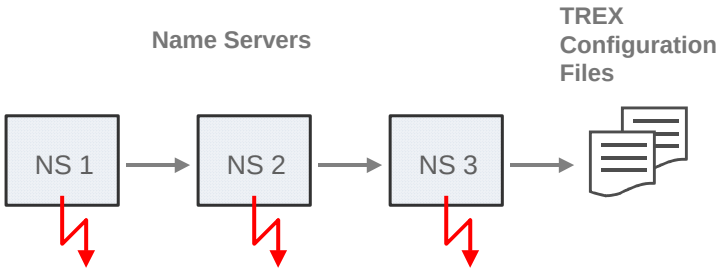


Communication then takes place using the alternative name server until it is stopped or goes down. When this happens, the next name server in the list takes over the communication role. This process continues as long as there are still entries in the name server list. When the end of the name server 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 operating a name server.
- TREX has been installed on the hosts involved (master and slaves).

Configuration

✔	**Action**
	On the name server:
	Set the master as the default index server [Page 64]
	On the master:
	Activate the Python extension handler [Page 71].
	Activate automatic export [Page 66]. This also defines whether all indexes or only selected indexes are exported.
	Stop the TREX daemon (see (UNIX) Starting and Stopping TREX [Page 96] and (Windows) Starting and Stopping the TREX Service [Page 100]).
	If you use query-based taxonomies; configure the TREX daemon on the master [Page 68].
	Start the TREX daemon (see (UNIX) Starting and Stopping TREX [Page 96] and (Windows) Starting and Stopping the TREX Service [Page 100]).
	Share the replication directory [Page 69] so that all slaves can access it. The replication directory contains the backup copies of the indexes.
	On all slaves:
	UNIX only: Mount the replication directory.
	Activate the Python extension handler [Page 71].
	Activate the slave extension [Page 71].

✔	**Action**
	Configure automatic import [Page 72]. This defines which indexes are imported and the time interval between imports.
	Stop the TREX daemon (see (UNIX) Starting and Stopping TREX [Page 96] and (Windows) Starting and Stopping the TREX Service [Page 100]).
	Configure the TREX daemon on the slaves [Page 79]
	Start the TREX daemon (see (UNIX) Starting and Stopping TREX [Page 96] and (Windows) Starting and Stopping the TREX Service [Page 100]).

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
	Set the master as the default index server	Start name server administration in the TREX directory: • Windows: `TREXNameServerGUI.py` • UNIX: `python TREXNameServerGUI.py` Method *setAsDefault* *Server Group* = **`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=search, thesaurus, admin`
		`preprocessor`
		`[extensionhandlers]`
		`trexxpy`

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

✔	Step	Summary
	Stop the TREX daemon	**On UNIX** As `trexadm`: `<TREX_Directory>/TREX stop` **On Windows** *Start → Programs → SAP TREX → TREX Service → stop*

	If you use query-based taxonomies: Configure the TREX daemon	Edit the `<TREX_Directory>\TREXDaemon.ini` file:
		`[daemon]`
		`programs = <existing_sections>, qbc`
		`[qbc]`
		`executable=python`
		`arguments=qbcrep.py --export`
		UNIX:
		`startdir=<TREX_Directory>/python_support/imp_exp_idx`
		Windows:
		`startdir=<TREX_Directory>\python_support\imp_exp_idx`
		`instances=1`
		`priority=normal`

	Start the TREX daemon	**On UNIX** As `trexadm`: `<TREX_Directory>/TREX start` **On Windows** *Start → Programs → SAP TREX → TREX Service → start*
	Share the replication directory	Share the directory `<TREX_Backup_Directory>\replication.`

On all slaves:

✔	Action	Summary
	UNIX only: Mount the replication directory	–

	Activate the Python extension handler	Edit the `<TREX_Directory>\TREXExtensions.ini` file:
		`[activate]`
		`imsapi=search, thesaurus, admin`
		`preprocessor`
		`[extensionhandlers]`
		`trexxpy`

	Activate the slave extension	Edit the `<TREX_Directory>\extensions.py` file: `# replication slave extension:` **`if 1`**

✔	Action	Summary
	Configure automatic import	Edit the `<TREX_Directory>\TREXImport.ini` file: `[autoimport]` **`sourcepath=<Replication_Directory>`** **`indexes=<Index1>,<Index2>,...`** **`active=yes`**
	Stop the TREX daemon	**On UNIX** As `trexadm`: `<TREX_Directory>/TREX stop` **On Windows** *Start → Programs → SAP TREX → TREX Service → stop*
	Configure the TREX daemon	Edit the `<TREX_Directory>\TREXDaemon.ini` file: `[daemon]` `programs = <existing_sections>,` **`importdaemon`** **`[importdaemon]`** **`executable=python`** **`arguments=import.py --daemon`** UNIX: **`startdir=<TREX_Directory>/python_support/imp_exp_idx`** Windows: **`startdir=<TREX_Directory>\python_support\imp_exp_idx`** **`instances=1`** **`priority=normal`** If you use query-based taxonomies: `[daemon]` `programs = <existing_sections>,` **`qbc`** **`[qbc]`** **`executable=python`** **`arguments=qbcrep.py`** UNIX: **`startdir=<TREX_Directory>/python_support/imp_exp_idx`** Windows: **`startdir=<TREX_Directory>\python_support\imp_exp_idx`** **`instances=1`** **`priority=normal`**
	Start the TREX daemon	**On UNIX** As `trexadm`: `<TREX_Directory>/TREX start` **On Windows** *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 among all known index servers.

For index replication, original indexes should be located only on the master. The name server should only forward create index requests to the master, not to the slaves. You must 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 daemon has started).

Procedure

1. Start name server administration (see Starting and Closing Name Server Administration [Page 112]).
2. Choose *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	Input
Server Group	**`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. Edit the configuration file `<TREX_Directory>\TREXExtensions.ini`.
2. Check that the `[activate]` section has the structure shown below, and modify it if necessary.

   ```
   [activate]
   imsapi=search, thesaurus, admin
   preprocessor
   ```

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

   ```
   [extensionhandlers]
   trexxpy
   ```

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. Edit the configuration file `<TREX_Directory>\extensions.py`.
2. Change the `if 0:` entry in the section that relates to the export of indexes to **`if 1:`**. You identify the section by the class name `ReplicationMaster`.
3. Check the `ReplicationMaster(<number-of-security-copies-per-index>,0)` call.

   The first parameter defines the number of security copies that are stored for each index. By default, the last two security copies 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'))

sys.path.append(os.path.join(os.getenv('SAP_RETRIEVAL_PATH'),
'python_support','imp_exp_idx'))
      from master import ReplicationMaster
      trexx.registerExtension(trexx.EXTCLASS_ADMIN,
ReplicationMaster(2,0))
```

4. Stop and restart the TREX daemon (see UNIX: Starting and Stopping the TREX Service [Page 96] or Windows: Starting and Stopping the TREX Service [Page 100]).

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. Edit the copied file. Make the changes described in the *Replicating All Indexes* section.
3. Stop and restart the TREX daemon (see UNIX: Starting and Stopping the TREX Service [Page 96] or Windows: Starting and Stopping the TREX Service [Page 100]).

Configuring the TREX Daemon on the Master

Use

If you use query-based taxonomies, a QBC replication script needs to run on the master and slaves.

This script enhances the functions of the standard replication procedure. It replicates the parts of query-based taxonomies that are not replicated by the standard procedure.

You should configure the TREX daemon on the master so that it starts the QBC replication script as well as the TREX servers.

Prerequisites

The TREX daemon has been stopped (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).

Procedure

1. Edit the configuration file `<TREX_Directory>\TREXDaemon.ini` on the master.
2. Create the following section:

   `[qbc]`

   `executable=python`

   `arguments=qbcrep.py --export`

   Windows: **`startdir=<TREX_Directory>\python_support\imp_exp_idx`**

   UNIX: **`startdir=<TREX_Directory>/python_support/imp_exp_idx`**

   `instances=1`

   `priority=normal`

   For information on the meanings of the parameters, see Configuring the TREX Demon [Page 29].
3. In the **`[daemon]`** section, in the `programs` parameter, add the **`qbc`** entry.

   `[daemon]`

   `programs=<existing_sections>,` **`qbc`**

Result

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

Sharing the Replication Directory

Use

The security copies 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 daemon has to have these access permissions.

Later on, you configure the TREX daemon so that it starts the import daemon. Thus, the import daemon runs on the same user as the TREX daemon.

- On UNIX, this is `trexadm` unless you chose another name for the TREX user. The TREX user is entered in the `<TREX_Directory>\TREXDaemon.ini` configuration file in the `[daemon]`, section, parameter `userid`.
- On Windows, this is the user that you defined for the TREX service.

By default, the backup directory is `<TREX_Directory>\backup`. If necessary, you can change the path in the configuration file (section `[Paths]`, parameter `backup`).

Procedure

Share the replication directory `<TREX_Backup_Directory>\replication` and all subdirectories. Give read and write permissions to the user that starts the import daemon.

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. Edit the configuration file `<TREX_Directory>\TREXExtensions.ini`.
2. Check that the `[activate]` section has the structure shown below, and modify it if necessary.

   ```
   [activate]
   imsapi=search, thesaurus, admin
   preprocessor
   ```

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

   ```
   [extensionhandlers]
   trexxpy
   ```

Activating the Slave Extension

Procedure

1. Edit the configuration file `<TREX_Directory>\extensions.py`.
2. Change the `if 0:` entry in the section that relates to the slave extension to **`if 1:`**. You identify the section by the class name `ReplicationSlave`.

   After the Change:

   ```
   # replication slave extension
     ...
     # ------------------------------
     if 1:
         sys.path.append(os.path.join(os.getenv('SAP_Retrieval_Path'),
   'extensions','replication'))
         sys.path.append(os.path.join(os.getenv('SAP_RETRIEVAL_PATH'),
   'python_support','imp_exp_idx'))
       from slave import ReplicationSlave
       trexx.registerExtension(trexx.EXTCLASS_ADMIN, ReplicationSlave(0))
   ```

Result

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

Configuring Automatic Import

Use

An import daemon must run on all slaves. The import daemon 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 daemon starts the import.

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

Prerequisites

- UNIX only: The replication directory has been mounted on the slaves.
- The indexes that are to be imported by the slaves have already been created on the masters.

Procedure

1. Edit the configuration file `<TREX_Directory>\TREXImport.ini` on the slave.
2. Make the desired changes (see TREXImport.ini Configuration File [Page 73]).

TREXImport.ini Configuration File

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

- The `[global]` section defines parameters that are valid for the import daemon.
- 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]
```

UNIX: `sourcepath=/<mountpoint_on_slave>`

Windows: `sourcepath=\\<hostname>\<share>`

```
indices=<index_ID1>,<index_ID2>
importevery=<minutes>
active=<yes|no>

[index_<index_ID>]
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 daemon. 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 daemon.

`sleeptime`

Automatic import.

Time in seconds that the daemon waits before becoming active again. After this amount of time, the daemon 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.

Replication directory of the master. The security copies of the original indexes are located in this directory.

UNIX: `sourcepath=/my_mount_point`

Windows: `sourcepath=\\indexserver1\replication`

`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 daemon 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 daemon 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 here for the parameter `importevery`. These settings can be different from the general settings in the `[autoimport]` section.

`importevery`

Automatic import.

Time in minutes for which the import daemon 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 daemon during routine operation.

`currentversiontimestamp`

Automatic import.

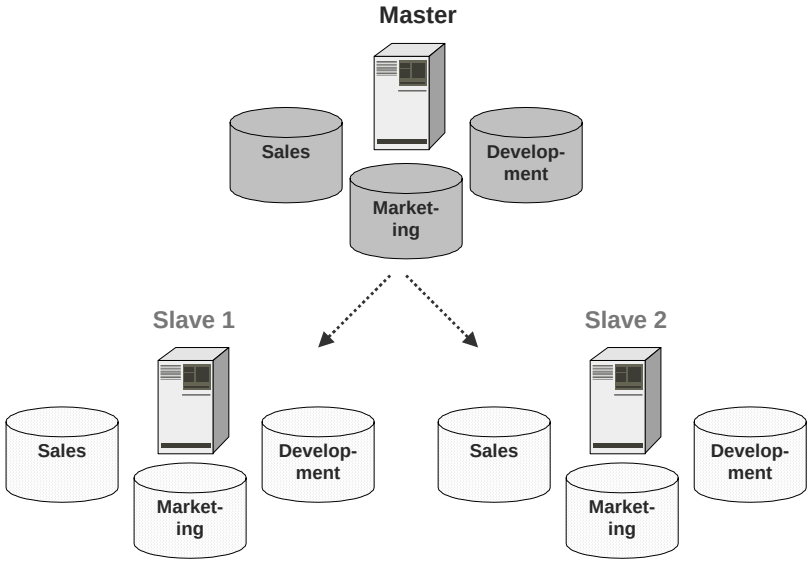
Time at which the last import process took place.

This parameter is filled by the import daemon 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 daemon 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]
```

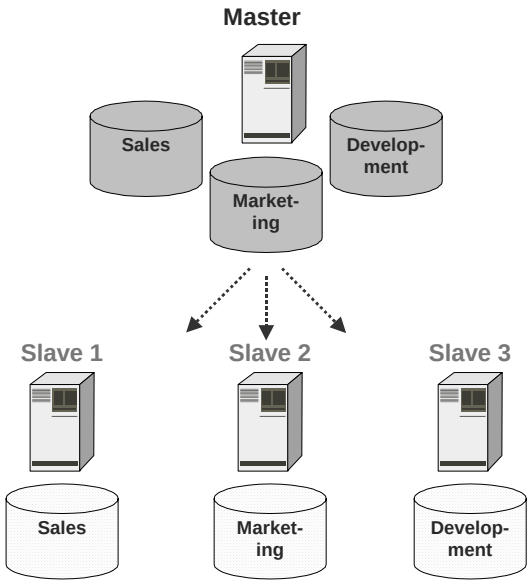
UNIX: **`sourcepath=/replication`**

Windows: **`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]
```

UNIX: `sourcepath=/replication`

Windows: `sourcepath=\\indexserver1\replication`

```
indices=sales
importevery=0
active=yes
...
```

Slave 2

```
[autoimport]
```

UNIX: `sourcepath=/replication`

Windows: `sourcepath=\\indexserver1\replication`

```
indices=marketing
importevery=0
active=yes
...
```

Slave 3

```
[autoimport]
```

UNIX: `sourcepath=/replication`

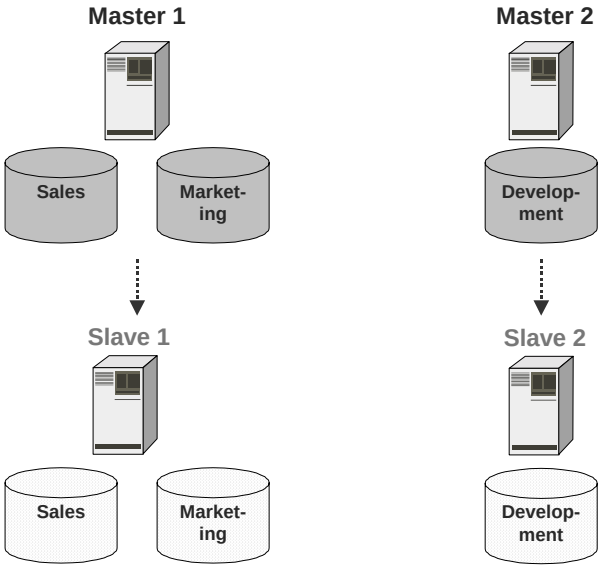
Windows: `sourcepath=\\indexserver1\replication`

`indices=development`

```
importevery=0
active=yes
...
```

Example 3

The graphic below depicts a scenario with two masters and two slaves. Slave 1 should import the indexes from master 1, and slave 2 those from master 2.



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

Slave 1

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

[autoimport]
```

UNIX: **`sourcepath=/replicationm1`**

Windows: `sourcepath=\\indexserver1\replication`

`indices=sales,marketing`

```
importevery=0
active=yes
```

Slave 2

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

[autoimport]
```

UNIX: **sourcepath=/replicationm2**

Windows: **sourcepath=\\indexserver2\replication**

indices=development

```
importevery=0
active=yes
```

Configuring the TREX Daemon on the Slaves

Use

You should configure the TREX daemon on the slaves so that is starts the following Python scripts as well as the TREX servers:

- The import daemon
- The QBC replication script (if required)

  This script is only required if you use query-based taxonomies.

Prerequisites

The TREX daemon has been stopped (see UNIX: Starting and Stopping TREX [Page 96] and Windows: Starting and Stopping the TREX Service [Page 100]).

Procedure

1. Edit the configuration file `<TREX_Directory>\TREXDaemon.ini` on the slaves.
2. Add the import daemon and the QBC replication script (if required).

Adding the Import Daemon

1. Create the following section:

   [importdaemon]

   executable=python

   arguments=import.py --daemon

   UNIX: **startdir=<TREX_Directory>/python_support/imp_exp_idx**

   Windows: **startdir=<TREX_Directory>\python_support\imp_exp_idx**

   instances=1

   priority=normal

   For information on the meanings of the parameters, see Configuring the TREX Demon [Page 29].

2. In the **[daemon]** section, in the `programs` parameter, add the `importdaemon` entry.

   ```
   [daemon]
   programs=<existing_sections>, importdaemon
   ```

Adding the QBC Replication Script

1. Create the following section:

   [qbc]

   executable=python

   arguments=qbcrep.py

   Windows: **startdir=<TREX_Directory>\python_support\imp_exp_idx**

   UNIX: **startdir=<TREX_Directory>/python_support/imp_exp_idx**

   instances=1

   priority=normal

2. In the **[daemon]** section, in the `programs` parameter, add the **qbc** entry.

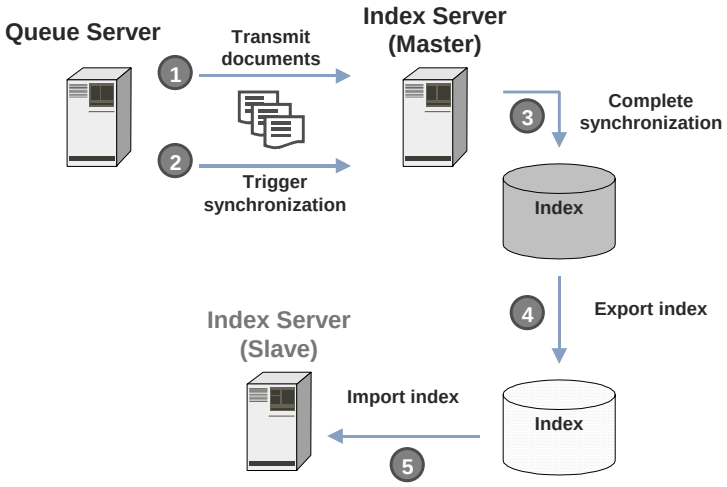
   ```
   [daemon]
   programs=<existing_sections>, qbc
   ```

Result

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

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 for Indexing

Purpose

The indexing of documents is a complex process in which all central TREX components are involved. One part of the indexing process is the preprocessing of documents. In this step, the preprocessor prepares the documents so that they can be forwarded to the index server later on. The actual indexing then takes place at the index server/on the TREX engines.

The preprocessing can take up a large part of the indexing time and need a similar number of system resources to the indexing itself. The preprocessing of a large number of documents that are not in text or HTML form can be particularly time- and resource-consuming (for example, large PDFs).

You can accelerate preprocessing by distributing it among more than one host. Distribution can also help you to increase data throughput when indexing take place.

The sections below explain how to implement distributed preprocessing.

- Overview [Page 83] explains the preprocessor's tasks and the preprocessing flow for indexing. It also explains how to distribute preprocessing and influence load distribution and performance.
- Implementing Distributed Preprocessing [Page 92] explains how to configure the preprocessors and queue server involved.
- Example Configuration [Page 94] shows the relevant sections from the TREX configuration files.

Overview

The following sections provide an overview of the topics below.

- Preprocessor Tasks
- Preprocessing Flow
- Distributed Preprocessing
- Load Distribution and Performance

Preprocessor Tasks

The preprocessor has the following tasks:

- Loading documents
- Filtering
- Carrying out a linguistic analysis

Loading documents

In the portal environment, TREX doesn't immediately receive the document to be processed itself. Instead, it receives a URL. This URI points to the repository in which the document is located. The preprocessor resolves the URI and loads the document from the repository using HTTP GET or HTTPS GET.

Filtering

Documents to be processed can have various formats such as Microsoft Word, Microsoft PowerPoint, or PDF. When filtering takes place, the preprocessor extracts the text content and makes it available for further processing. The text content then exists in the form of an HTML document encoded in Unicode format UTF-8.

Carrying out a linguistic analysis

The preprocessor can linguistically analyze documents and search queries. A linguistic analysis consists of the following steps:

- Language recognition
- Determination of words and sentence limits (tokenization)
- Unification of spelling (normalization)
- Determination of word types (tagging)
- Reduction of grammatically related words to a single root form

  Example: The words 'computing' and 'computers' are reduced to the root form 'compute'.
- Removal of stop words (for example, the articles 'the' and 'a').

The preprocessor uses a lexicon that is available in several languages to do this.

After the linguistic analysis has taken place, the document is no longer available as readable text. All information is then encoded in a special format called 'Document Analysis Format' (DAF).

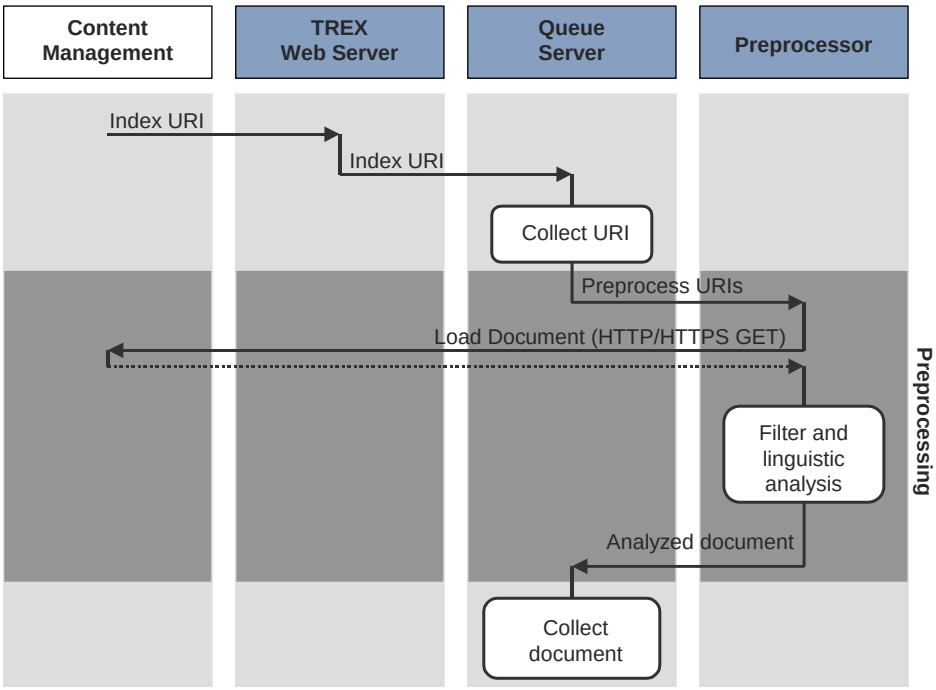
In which processes does the preprocessor play a part?

The preprocessor is involved in all central TREX processes – not just indexing, but also the preprocessing of search and text-mining requests. In all of these processes, the preprocessor has the task of preparing the actual preprocessing.

The sections below only relate to the preprocessing of documents for indexing. The preprocessing of search queries and the preprocessing of documents when answering text-mining requests are not covered.

Preprocessing Flow

The figure below depicts the most important steps that take place before, during, and after preprocessing. It applies only to the portal environment. To keep the figure as simple as possible, the name server and communication with the name server are not depicted.



Content Management forwards the document to be indexed to the TREX Web server in the form of a URI. The Web server then forwards the URI to the queue server. The queue server collects URIs and distributes them to one or more preprocessors. The preprocessing of documents then takes place at the preprocessor(s).

When the preprocessing has been completed, the preprocessor passes the analyzed document to the queue server. The queue server collects documents and forwards them to the index server according to the configured queue parameters. The actual indexing then takes place at the index server/in the TREX engines.

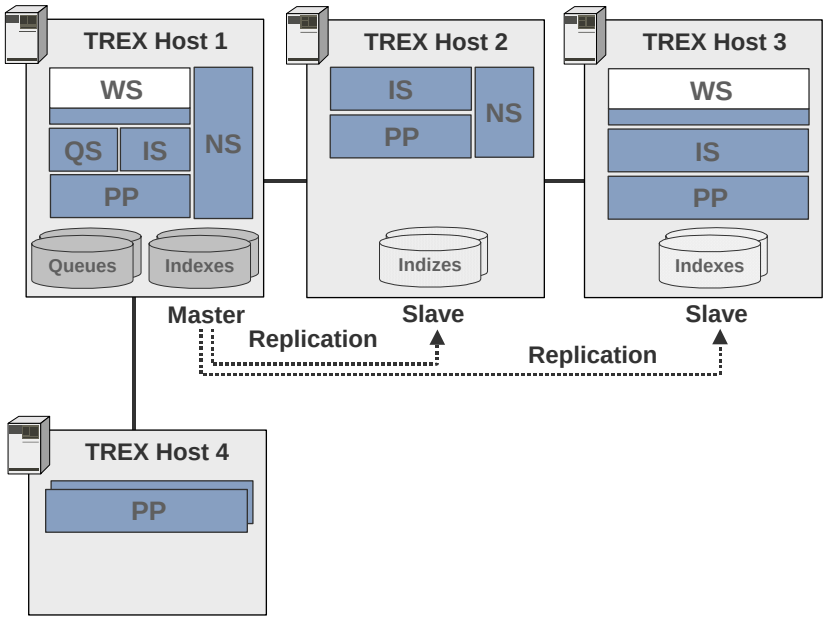
Distribution with Several Hosts

Preprocessing normally takes place on one host. If you implement distributed preprocessing, an additional host preprocesses documents exclusively. Only the preprocessor runs on this host. The queue server can then distribute the indexing requests among several preprocesses, thereby allowing TREX to preprocess a greater number of documents in parallel.

Example

The figure below shows documents being preprocessed on two hosts (hosts 1 and 4). Host 1 is responsible for the entire indexing process (collecting, preprocessing, and indexing documents) and for processing search requests. Host 4 is responsible exclusively for preprocessing.

For a detailed description of the example, see Error-Tolerant and Scalable System [Page 15].



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

The preprocessors on hosts 2 and 3 are not responsible for preprocessing for indexing. Instead, they preprocess search and text-mining requests.

Load Distribution and Performance

If you are implementing distributed preprocessing, you have to configure the preprocessors involved. The following parameters denote load distribution and performance for preprocessing.

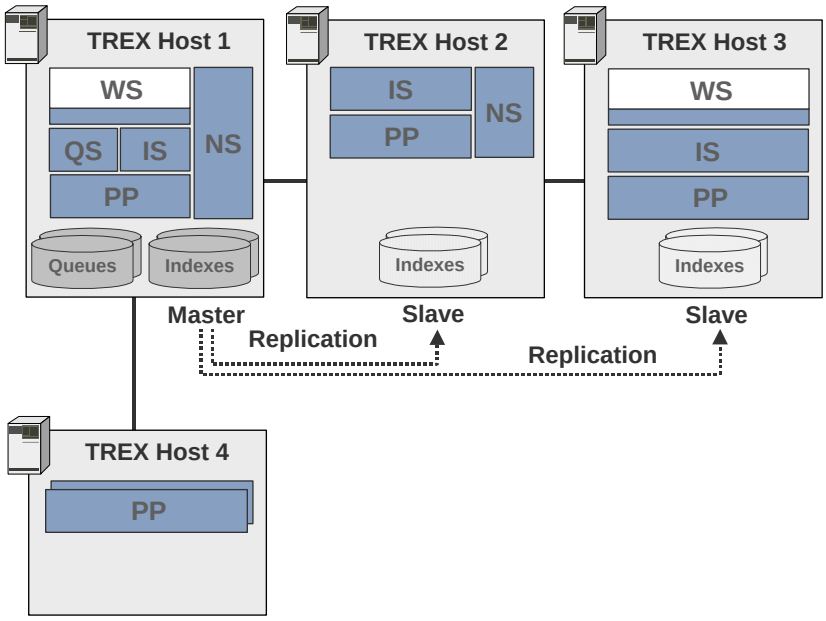
- Preprocessor instances per host
- Preprocessor threads
- Pool size of queue server

Multiple Preprocessor Instances per Host

If a host only preprocesses documents for indexing, you may want to run more than one preprocessor instance on it. This allows you to control load-distribution on the hosts and increase breakdown backup.

Load Distribution

The figure below depicts preprocessing distributed among two hosts (host 1 and host 4). Host 1 indexes and processes search queries as well as preprocessing documents. It should therefore receive a lesser load for preprocessing than host 4. This is why two instances of the preprocessor are running on host 4. The load is distributed among the hosts in the ration 1:2.



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

Breakdown Backup

If you have more than one instance on the same host, you increase breakdown backup for the system. This is because different processes (instances) have less impact on one another than do the different threads of a process. If a thread hangs, this can affect other threads of the same process but not of another process.

For information on instances and threads as regards performance, see Preprocessor Threads [Page 89].

Number of Instances

The number of preprocessor instances that can run per host depends on the number of CPUs and on the available memory space.

We recommend the following maximum values:

Number of instances = number of CPUs + (1 or 2)

The main memory requirement per instance depends on the size of the documents, their format (PDF, HTML, and so on) and the number of languages to be recognized. With **one** language, the main memory requirement is between 30 and 40 MB per instance. For several languages, it can be in the region of 100 MB per instance.

In less than ideal cases, the main memory requirement per instance can be in the region of 500 MB, and even reach 2 GB in the worst case scenario. The worst case scenario can occur if language recognition is activated for all languages and a large number of preprocessor threads are processing large documents at the same time.

Preprocessor threads

Each preprocessor instance is a separate process that consists of at least one thread. If the process has multiple threads, requests to the threads can be distributed so that the system can process multiple requests in parallel.

The threads of a preprocessor are divided into several classes. Each class is responsible for particular requests as shown below.

Thread Class	Description
`workerthreads_0`	Threads for preprocessing documents
`workerthreads_1`	Threads for preprocessing search requests
`workerthreads_2`	Not currently used

The class `workerthreads_0` is important for the performance of preprocessing for indexing. The higher the number of these threads, the more system resources are devoted to the preprocessor. This in turn determines the number of documents that can be preprocessed in parallel. The maximum number of threads possible depends on the setup of the host.

If you want the preprocessor to have as many system resources as possible and to use an almost complete CPU load, we recommend the following value:

`workerthreads_0` = number of CPUs * (3 or 4)

Therefore, if a host has 2 CPUs, the number of threads can be between 6 and 8.

If you want the preprocessor to have fewer system resources, you can choose to have a smaller number of threads. However, you ought not choose to have a greater number of threads, since this can lead performance to drop.

The more threads invoked in parallel, the longer the operating system takes to administrate the threads (to trigger, stop, and monitor them). If the number of threads invoked in parallel is too great, the operating system is overwhelmed by thread administration.

Preprocessor Instances and Threads

There is no performance benefit to be gained by having multiple instances with fewer threads as opposed to one instance with a greater number of threads.

You start with multiple instances to increase system resources for the host. You start multiple threads to increase performance.

In certain situations, multiple instances can also lead to performance benefits. For example, when processing a large number of documents, the system takes a long time filtering them. If only one instance is running, it might be that its threads are busy but there are still free system resources. As the maximum recommended number of threads has been reached and the instance has no more free threads, it can no longer receive documents. The host can only process additional documents if you add another instance. This instance can then receive additional documents and distribute them among its threads.

Pool Size of Queue Server

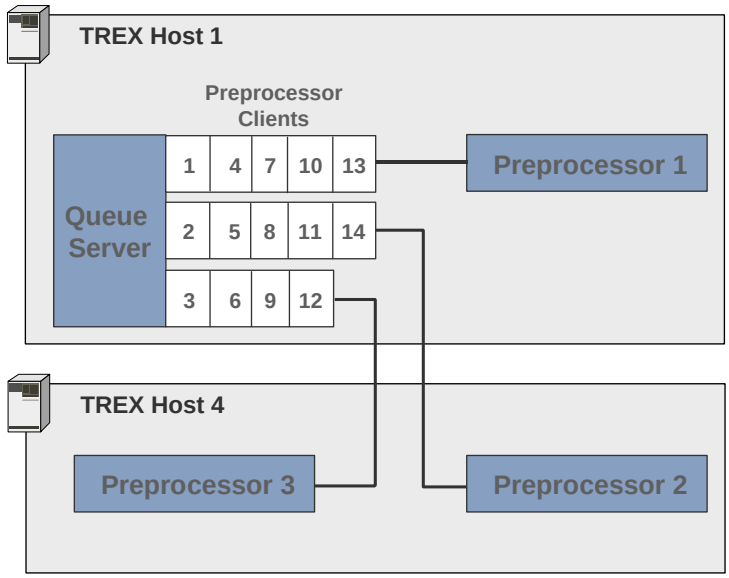
The pool size is important for achieving optimum integration between the queue server and preprocessors. The pool size specifies how many preprocessor clients the queue server instantiates when it starts. The queue server uses the preprocessor clients to communicate with the preprocessors and users its services. The pool size also determines how many documents the queue server can distribute to the preprocessors at once.

How does the queue server distribute documents?

When the queue server starts, it instantiates as many preprocessor clients as prescribed by the pool size. The queue server shares the pool size between the number of preprocessors recognized by it, and assigns a fixed number of preprocessor clients to each preprocessor. The assignment of preprocessor clients to preprocessors is fixed and does not change during the runtime of the queue server.

Example

If you have a pool size of 14 and 3 preprocessors, assignment is as depicted below.



When the queue server receives the documents to be indexed, it distributes them among the preprocessor clients. The clients then forward the documents to the preprocessors and wait for their response. Clients are busy while waiting for a response. They receive no further documents from the queue server during this time. When the preprocessing of the documents is over, the client receives a response from the preprocessor, and returns its own response to the queue server. Only then is the client free to receive further documents from the queue server.

What value should the pool size have?

You should select the pool size that allows the optimum load for the preprocessors.

We recommend the following initial values:

Pool size = total of `workerthreads_0` from all hosts preprocessing documents for indexing + (1 or 2).

If the pool size is smaller than this, the preprocessors may have unnecessary idle time in between processing documents.

You have to determine the optimal pool size for your particular system. Start by using the recommended value, and monitor the CPU load for the preprocessors for a while. If system resources are still available, you can increase the pool size to improve performance. However, if you increase the pool size too much, you gain no performance benefits and might actually cause performance to drop.

Example

Preprocessing is distributed among two hosts. One preprocessor instance is running on host 1, and two instances are running on host 4. The parameter `workerthreads_0` has the value 6 on both hosts (for the calculation of this value, see Preprocessor Threads [Page 89]).

This gives rise to the following pool size:

`workerthreads_0` on host 1 + `workerthreads_0` on host 2 + (1 or 2) = 13 or 14

If several preprocessor instances are running on one host, the `workerthreads_0` count once per host, not once per instance. Therefore, 6 threads are counted for host 4 instead of 12.

Implementing Distributed Preprocessing

Use

The following section describes how you configure the preprocessors and queue server for distributed preprocessing. The description assumes that

- You are already operating one TREX host.
- You want to implement an additional TREX host that only has the task of preprocessing documents for indexing.

Prerequisites

- TREX is installed on the new host.
- The TREX daemon has already been configured on the new host, so that it starts either one or multiple preprocessor instances (see Configuring the TREX Daemon [Page 29] and Example Configuration for a Large Distributed System [Page 35]).

Procedure

On the new host

1. Stop the TREX daemon (see UNIX: Starting and Stopping TREX [Page 96] or Windows: Starting and Stopping the TREX Service [Page 100]).
2. If necessary, change the parameter `workerthreads_0` in the configuration file `<TREX_Directory>\TREXPreprocessor.ini`.

   ```
   [service]
   workerthreads_0 = <Number of Threads>
   ```

   For information on the number of threads, see Preprocessor Threads [Page 89].

On the host on which the queue server is running

1. Stop the TREX daemon.
2. If necessary, change the parameter `workerthreads_0` in the configuration file `<TREX_Directory>\TREXPreprocessor.ini`.

   ```
   [service]
   workerthreads_0 = <Number of Threads>
   ```

   For information on the number of threads, see Preprocessor Threads [Page 89].

   Change the following parameters in the configuration file `<TREX_Directory>\TREXQueueServer.ini`:

Parameters in [preprocessor] section	**Description**
`poolsize`	Number of preprocessor clients that the queue server instantiates. This relates to the number of documents that the queue server is able to distribute to the preprocessors at once. For information on calculating the pool size, see Pool Size of Queue Server [Page 90].
`preprocessors`	Number of preprocessors among which the queue server can distribute documents.
`host<n>` `port<n>`	Addresses of preprocessors (hostname and port) among which the queue server can distribute documents. Number the addresses sequentially.  The number of the addresses must be the same as the number of preprocessors (parameter `preprocessors`).

On both hosts

Start the TREX daemon (see UNIX: Starting and Stopping TREX [Page 96] or Windows: Starting and Stopping the TREX Service [Page 100]).

See also:

Example Configuration [Page 94]

Example Configuration

Below are the sections from the TREX configuration files that are relevant for distributed preprocessing.

The purpose of this configuration is to give as many system resources as possible for preprocessing for indexing. You can check whether the configuration is optimal by checking the CPU usage on the hosts involved. When documents are being preprocessed, the CPU usage should always be at the upper limit.

The configuration relates to two hosts with two CPUs each.

Configuration of Host 4

TREXDaemon.ini (extract)

```
[daemon]
programs = preprocessor1, preprocessor2

[preprocessor1]
executable=TREXPreprocessor
arguments=-port 8357
```

UNIX: **startdir=/my_path/my_TREX_directory**

Windows: **startdir=D:\my_path\my_TREX_directory**

```
instances=1
priority=normal

[preprocessor2]
executable=TREXPreprocessor
arguments=-port 8359
```

UNIX: **startdir=/my_path/my_TREX_directory**

Windows: **startdir=D:\my_path\my_TREX_directory**

```
instances=1
priority=normal
```

TREXPreprocessor.ini (extract)

```
[service]
workerthreads_0 = 6
```

Configuration of Host 1

TREXPreprocessor.ini (extract)

```
[service]
workerthreads_0 = 6
```

TREXQueueServer.ini (extract)

```
[preprocessor]
poolsize=14
preprocessors=3
host1 = thisHost
port1 = 8357
host2 = anotherHost
port2 = 8357
host3 = anotherHost
port3 = 8359
```

In this configuration, the queue server distributes documents to three preprocessors. One preprocessor instance runs on the same host as the queue server. The other instances run on host 4.

Calculation of the pool size: `workerthreads_0` on host 4 + `workerthreads_0` on host 1 + 2 = 14.

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 TREX on UNIX

Purpose

The following sections explain how to start and stop TREX on UNIX.

In the TREX installation directory, there is a `TREX` script that stops and starts the TREX daemon. During the installation you made sure that TREX would be started and stopped automatically. You can also call up the `TREX` script manually, for example, if you want to stop TREX temporarily and then restart it.

For test purposes and troubleshooting, you can also

- Start and stop the TREX server and Web server individually
- Stop and start the TREX daemon in debug mode

Starting and Stopping TREX

Use

You can start and stop the TREX daemon manually by calling up the `TREX` script with the appropriate option.

Starting TREX

1. Log on with the user `trexadm`.
2. Go to the TREX installation directory, and call up the `TREX` script using the `start` option.

   ```
   cd <TREX_Installation_Directory>
   TREX start
   ```

Stopping TREX

1. Log on with the user `trexadm`.
2. Go to the TREX installation directory, and call up the `TREX` script using the `stop` option.

   `cd <TREX_Installation_Directory>`

   `TREX stop`

   Do not stop the TREX daemon using `kill –9`, and do not stop the individual child processes that the daemon has started. Doing so can lead to the loss of data. Affected indexes can be irreparably damaged.

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

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

Starting and Stopping Individual TREX Servers

Use

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. Log on with the TREX user (normally `trexadm`).
2. Stop the TREX daemon (see Starting and Stopping TREX [Page 96]).
3. Go to the TREX directory.
4. Select the TREX servers that you need. Start each server in a separate shell.
   - If you have a non-distributed installation, 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.x`**
Preprocessor	**`TREXPreprocessor.x`**
Queue server	**`TREXQueueServer.x`**
Only for a distributed system: Name server	**`TREXNameServer.x`**

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 servers.

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

Do not stop the TREX servers using `kill – 9`, as this can lead to data loss. Affected indexes can be irreparably damaged.

Starting and Stopping the TREX Daemon in Debug Mode

Use

If you start the TREX daemon in debug mode, you can track the program output on the screen. You do this for test purposes and troubleshooting.

Starting the TREX Daemon

1. Log on with the TREX user (normally `trexadm`).
2. Stop the TREX daemon (see Starting and Stopping TREX [Page 96]).
3. Open a shell and go to the TREX directory.

   `cd <TREX_Directory>`
4. Execute the following command:

   `TREXDaemon.x -d`

Stopping the TREX Daemon

1. Display the window in which you started the TREX daemon.
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 daemon is stopped. Therefore, it can take a certain amount of time to stop the daemon.

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

Do not stop the TREX daemon using `kill – 9`, as this can lead to data loss. Affected indexes can be irreparably damaged.

Starting and Stopping the Web Server

Use

You can start and stop the Apache Web server manually if necessary.

Starting the Web Server

1. Log on with the TREX user (normally `trexadm`).
2. Go to the Apache installation directory, and call up the `apachectl` script using the `start` option.

   `cd <TREX_Directory>/Apache`

   `apachectl start`

Stopping the Web Server

1. Log on with the TREX user (normally `trexadm`).
2. Go to the Apache installation directory, and call up the `apachectl` script using the `stop` option.

   `cd <TREX_Directory>/Apache`

   `apachectl stop`

Starting and Stopping TREX on Windows

Purpose

The following sections explain how to start and stop TREX on Windows.

The TREX setup registers the TREX daemon as a service. The TREX service is configured so that it starts automatically when the host is started up, and stops automatically when the host is shut down. If the TREX service and the Web server are running, TREX is ready to use.

You can also start and stop the TREX service and Web server manually. For test purposes or troubleshooting, you can also:

- Stop and start the TREX servers individually
- Stop and start the TREX daemon in debug mode

Starting and Stopping the TREX Service

Use

The TREX setup registers the TREX daemon as a service. The TREX service is configured so that it starts automatically when the host is started up, and stops automatically when the host is shut down.

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

Prerequisites

You restarted the host after you installed TREX.

Starting the TREX Service Manually

Choose:

- Windows 2000: *Start* → *Programs* → *SAP TREX* → *TREX Service* → *start*.
- Windows Server 2003: *Start* → *All Programs* → *SAP TREX* → *TREX Service* → *start*.

Stopping the TREX Service Manually

Choose:

- Windows 2000: *Start* → *Programs* → *SAP TREX* → *TREX Service* → *stop*.
- Windows Server 2003: *Start* → *All Programs* → *SAP TREX* → *TREX Service* → *stop*.

Certain processing steps, for example, writing an index, cannot be interrupted. Such steps are completed before the TREX service is stopped. This process can take a while to complete.

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

Starting and Stopping Individual TREX Servers

Use

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 100]).
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 servers.
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 servers.

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

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

Starting and Stopping the TREX Daemon in Debug Mode

Use

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

Starting the TREX Daemon

1. Stop the TREX service (see Starting and Stopping the TREX Service [Page 100]).
2. Open a prompt and go to the TREX installation directory.
3. Execute the following command:

   ```
   TREXDaemon.exe -d
   ```

Stopping the TREX Daemon

1. Display the window in which you started the TREX daemon.
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 daemon is stopped. Therefore, it can take a certain amount of time to stop the daemon.

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

Do not use the Task Manager to stop the TREX daemon. 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 IIS) manually.

Starting the Web Server

1. Choose:
   - Windows 2000: *Start* → *Settings* → *Control Panel* → *Administrative Tools* → *Services*.
   - Windows Server 2003: *Start* → *Administrative Tools* → *Services*.
2. Select *IIS Admin Service* and choose *Start* from the context menu.

If the World Wide Web publishing service doesn't run even though you have started it, try to start it using a prompt.

1. Open a 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.

Microsoft IIS 5.0

1. Choose:
   - Windows 2000: *Start* → *Programs* → *Administrative Tools* → *Internet Services Manager*.
   - Windows Server 2003: *Start* → *Administrative Tools* → *Internet Services Manager*.
2. Select the local host and choose *Action* → *Restart IIS*.

Microsoft IIS 6.0

1. Choose:
   - Windows 2000: *Start* → *Programs* → *Administrative Tools* → *Internet Information Services (IIS) Manager*.
   - Windows Server 2003: *Start* → *Administrative Tools* → *Internet Information Services (IIS) Manager*.
2. Select the local host and choose *Action* → *All Tasks* → *Restart IIS*.

Stopping the Web Server

1. Choose:
   - Windows 2000: Choose *Start* → *Settings* → *Control Panel* → *Administrative Tools* → *Services*.
   - Windows Server 2003: *Start* → *Administrative Tools* → *Services*.
2. Select *IIS Admin Service* and 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:

   Windows: **`deleteIndex.py <index_ID>`**

   UNIX: **`python deleteIndex.py <index_ID>`**

   Example: (UNIX): `python deleteIndex.py marketing`

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

Deleting the Entry from the Name Server Databasis

1. Start name server administration (see Starting and Closing Name Server Administration [Page 112]).
2. Choose *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	Input
Server Group	**`indexserver`**
Address	Address of the slave from which you have deleted the index copy. **`<hostname>:<port>`**
Entry	**`index_<index-ID>`**

   Server Group: **`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 28]).
2. Register the name server in the `TREXConfigMgr.ini` configuration file (see Registering the Name Server with All Remaining TREX Servers [Page 42]).
3. Carry out all steps necessary for index replication on a slave (see Implementing Index Replication [Page 59]).
4. Start TREX on the new host (see (UNIX) Starting and Stopping TREX [Page 96] and (Windows) Starting and Stopping the TREX Service [Page 100]).
5. Start name server administration (see Starting and Closing Name Server Administration [Page 112]). Check whether the new index server is recognized by the name server and is marked as active.

   Use the `listGroupEntries` method and enter `indexserver` as the server group. 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 could not be deleted from all slaves [Page 104], section *Deleting the Index Copy from the Slave*).
2. Delete the index entry from the name server databasis (see A replicated index could not be deleted from all slaves [Page 104], 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 have to stop TREX on the slave (see (UNIX) Starting and Stopping TREX [Page 96] and (Windows) Starting and Stopping the TREX Service [Page 100]).

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.

Proceed as follows:

On the slave

1. Stop TREX (see (UNIX) Starting and Stopping TREX [Page 96] and (Windows) Starting and Stopping the TREX Service [Page 100]).
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 daemon and, if it exists, the section for the QBC replication script
   - In the `[daemon]` section, in the `programs` parameter, delete the `importdaemon` entry and, if it exists, the entry for the QBC replication script
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'))
    sys.path.append(os.path.join(os.getenv('SAP_RETRIEVAL_PATH'),
'python_support','imp_exp_idx'))
    from slave import ReplicationSlave
    ...
```

On the name server

1. Start name server administration (see Starting and Closing Name Server Administration [Page 112]).
2. Identify the name server using *Address*.
3. Choose *deleteService.*
4. Enter the address of the slave, for example, **`mytrexhost2:8351`**.
5. Choose *Send.*

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.

Appendix: Name Server Administration

Purpose

Name server administration is a tool that you can use to query and change the name server databasis. The databasis contains information on all TREX servers known to the name server. The databasis also contains information on which indexes or queues are located on which server.

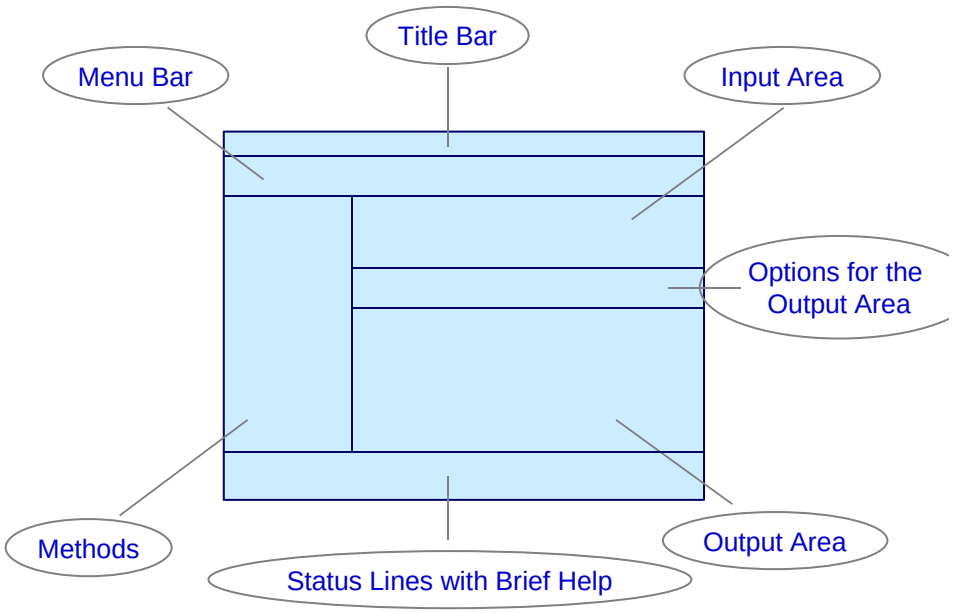
You can use name server administration to

- Activate the watchdog function of the name server
- Activate additional name servers that are to participate in name server replication
- Indicate that the master is the default index server for index replication
- Clean up the name server databasis in case of errors

Features

User Interface

The user interface of name server administration consists of the following areas:



Methods

Several functions are available for querying and changing the name server databasis. These are displayed on the user interface as methods. The table below gives an overview of the methods.

Method	**Description**
getServInfo	Displays the address of a server on which an index/queue is located
getVersion	Displays the name server version
listAllGroups	Displays the server groups known to the name server
showAllMembers	Displays all information that the name server databasis contains about individual servers
isEntryKnown	Checks whether an index or queue is entered in the name server databasis
listGroupEntries	Displays which indexes or queues are located on which server
putServData	Changes data entered for a server in the name server databasis, or adds new data
delEntryData	Deletes an index/queue from the name server databasis
setInactive	Marks a server as inactive
setActive	Marks a server as active
setAsDefault	Marks a server as the default server
reloadData	Causes the name server to reload its databasis from the hard drive
saveData	Causes the name server to save its databasis on the hard drive
watchdog	Changes the status of the watchdog or returns its current status
getAddressListForEntry	Displays the addresses of the servers on which an index or queue is located
getEntryInfo	Displays the parameters of an index or queue in the name server databasis
deleteService	Deletes a server from the name server databasis
In the *Advanced* menu	
Test Replicated Name Servers	Tests whether the replication of the name server databasis is functioning
Auto Repair Replicated Name Servers	Repairs inconsistencies in the name server databases

Administration Session

Purpose

You can query and change the name server databasis in an administration session.

Prerequisites

The name server whose databasis you want to access is running.

Process Flow

1. Start name server administration.
2. Specify the name server whose databasis you want to access.
3. Carry out the required methods.

Starting and Closing Name Server Administration

Use

You can start name server administration on any TREX host, regardless of where the name server is running.

Starting Name Server Administration

On UNIX

Go to the TREX directory and start the script `TREXNameServerGUI.py`:

```
cd <TREX_Directory>
python TREXNameServerGUI.py
```

On Windows

Start the following Python script by double-clicking on it with the primary mouse button:

`<TREX_Directory>\TREXNameServerGUI.py`

Closing Name Server Administration

Use the primary mouse button to click on X (*Close*) in the title bar.

Specifying the Name Server

Use

You specify the name server before executing a method.

Prerequisites

The name server whose databasis you want to access is running.

Procedure

1. Choose *Address* and enter the address of the name server:

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

   Unless you specified another port during the installation of TREX, the port number is 8355.

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

2. Choose *OK.*

Result

You should now test that the name server can be reached. For example, execute *getVersion* and check that the name server version is displayed.

Executing Methods

Prerequisites

- The name server whose databasis you want to access is running.
- You have specified the name server (see Specifying the Name Server [Page 113]).

Procedure

1. Choose the required method.
2. Enter the required data into the input area. Mandatory fields are indicated by an asterisk (*).

   If you have to enter the server group (for example, index server, queue server) for a method, you can select the group from a selection list. You display the selection list by clicking with the primary mouse-button on the question mark (?).

3. When you have entered all the data, choose *Send*.

Result

The output area displays the response of the name server.

By default, the output area contains the responses to all methods that you executed in one administration session. You can change the display in the output area as follows:

- Choose *Clear* to delete all responses of the name server.
- Choose *Show only last response* to display only the response to the last method executed. This option takes effect when you execute the next method.

Method References

The following sections contain a description of the methods available in name server administration. The methods are described in the order in which they appear on the user interface. There is the following information on each method:

- What is the purpose of this method?
- Which entries are required?
- What output do you receive when you execute the method?
- What are typical examples (input and output) for this method?

getServInfo

Use

This method displays the address of a server on which an index or queue is located.

The method only displays the address of **one** server. If multiple servers are possible, one of the servers is selected in a round-robin procedure.

Features

Input

Field	Description
Server Group	Group to which the server belongs. For this method, only `indexserver` or `queueserver` are sensible.
Entry	Index or queue in the format `index_<index_ID>`.
Query	Optional One or more parameters in the format `<parameter_name>=<value>` You can specify all parameters that are defined for the index or queue. For more information, see Name Server Databasis [Page 134], section [index_<index_ID>].

Output

If the relevant index or queue exists, the address of one of the servers on which the index or queue in question is located is displayed. Otherwise, you receive a message telling you that no matching data was found.

Example

On which server is the index ‘marketing’ located?

Input

Server Group: **indexserver**

Entry: **index_marketing**

Query:

Output

```
OK – mytrexhost1:8351
```

On which server is the queue for the index ‘marketing’ located?

Input

Server Group: **queueserver**

Entry: **index_marketing**

Query:

Output

```
OK – mytrexhost1:8351
```

On which server is the index ‘marketing’ with the parameter is_writable=no located?

Input

Server Group: **indexserver**

Entry: **index_marketing**

Query: **is_writable=no**

Output

If an index with this parameter exists:

```
OK – mytrexhost2:8351
```

If no index with this parameter exists:

```
No data found
```

getVersion

Use

This method displays the name server version. You can use this method to check the following:

- Did you specify the address of the name server correctly and can the name server be reached?
- Is the name server available for use?

Features

Input

No input required.

Output

If the name server can be reached and is available for use, the name server version is displayed. If not, you receive a message telling you that the name server cannot be reached.

listAllGroups

Use

This method displays the server groups known to the name server.

Features

Input

No input required.

Output

The server groups are displayed in the following format:

```
httpserver
indexserver
nameserver
preprocessor
queueserver
```

showAllMembers

Use

This method displays all information that the name server databasis contains about individual servers. This allows you to check, for example, whether a server is recognized by the name server, and the data that is stored on the server in question.

If you execute this method without further entries, the entire content of the name server databasis is displayed. You can also restrict the display to a server group (for example, only display data for index servers) or to an individual server.

Features

Input

Field	Description
Server Group	Optional Group of the server for which you want to display information; `indexserver`, `queueserver`, `httpserver`, `nameserver`, or `preprocessor`

Address	Optional
	Address of the server for which you want to display information. The address must have the following format:
	• Current name server
	`__ME__`
	The parameter `__ME__` has two underscores before `ME` and two underscores after it.
	• Web server
	UNIX: **`http://<hostname>:<port>/trex`**
	Windows: **`http://<hostname>:<port>/trexhttpserver/trexhttpserver.dll`**
	The standard port is 8353.
	• Other servers
	`<hostname>:<port>`
	Standard ports:
	○ Index server 8351
	○ Name server 8355
	○ Preprocessor 8357
	○ Queue server 8352

Output

The name server databasis is displayed in its entirety or in excerpts. For more information on the structure of the name server databasis, see Name Server Databasis [Page 134].

Example

What content does the name server databasis have?

Input

Server Group:

Address:

Output

The name server databasis is displayed in its entirety.

What information does the name server databasis contain on the index servers?

Input

Server Group: `indexserver`

Address:

Output

Only information on the index servers is displayed. In particular, you can see which indexes are located on which index servers, and which parameters the indexes in the name server databasis have.

What information does the name server databasis contain on a particular index server?

Input

Server Group: `indexserver`

Address: `mytrexhost1:8351`

or

Server Group:

Address: `mytrexhost1:8351`

Output

Only information on the specified index server is displayed. In particular, you can see which indexes are located on the index server, and which parameters the indexes in the name server databasis have.

isEntryKnown

Use

This method checks whether an index or queue is entered in the name server databasis.

Features

Input

Field	Description
Server Group	Optional Group of servers to which the entry belongs. For this method, only `indexserver` or `queueserver` are sensible.
Entry	Entry whose availability you want to check. Make the entry in the format `index_<index_ID>`.

Output

If the entry is available, `isEntryKnown=true` is displayed. If not, `isEntryKnown=false` is displayed.

Example

Is the index or queue ‘marketing’ available?

Input

Server Group:

Entry: **`index_marketing`**

Output

If the index or queue is available:

`isEntryKnown=true`

Is the index ‘marketing’ available?

Input

Server Group: **`indexserver`**

Entry: **`index_marketing`**

Output

If the index is available:

`isEntryKnown=true`

Is a queue for the index ‘marketing’ available?

Input

Server Group: **queueserver**

Entry: **index_marketing**

Output

If the queue is available:

```
isEntryKnown=true
```

listGroupEntries

Use

This method displays which indexes or queues are located on which server. This method also displays whether a server is active and how often the name server has forwarded the address of a server.

Features

Input

Field	Description
Server Group	Group of servers for which you want to display information; `indexserver`, `queueserver`, `httpserver`, `nameserver`, or `preprocessor`

Example

Which indexes are located on the index servers?

Input

Server Group: **indexserver**

Output

```
indexserver=mytrexhost1:8351
isActive=true
accessCount=2150
[index_marketing]
[index_sales]
[index_development]

indexserver=mytrexhost2:8351
isActive=true
accessCount=2120
[index_marketing]
[index_sales]
```

Which queues are located on the queue server?

Input

Server Group: **queueserver**

Output

```
queueserver=mytrexhost1:8352
isActive=true
accessCount=2150
[index_marketing]
[index_sales]
[index_development]
```

putServData

Use

This method changes data entered for a server in the name server databasis, or adds new data

You use this method mainly to activate the watchdog permanently. You can also use this method to

- Deactivate an index temporarily.
- Stipulate that the master indexes should not be used for searching when implementing index replication.

Only use the method for the latter two purposes after consulting with TREX Support.

Features

Input

Field	Description
Server Group	Group of servers for which you want to display or add data; `indexserver`, `queueserver`, `httpserver`, `nameserver`, or `preprocessor`
Address	Address of the server. For the format of the address, see showAllMembers [Page 117].

Data	Optional
	Data that you want to change or add. Enter the data in the following format:
	`[<entry_name>]`
	`<parameter_name1>=<value1>`
	`<parameter_name2>=<value2>`
	`...`
	 
	The spelling of your entries is not checked. The validity of the specified entry name and parameters are also not checked. Check your entries again before executing this method.

Output

The output area displays your entries again. The message `OK – putServData` tells you that the data was changed or added.

Example

Activating the Watchdog Permanently

Input

Server Group: **`indexserver`**

Address: **`__ME__`**

Data:

```
[_watchdog]
isActive=true
interval=10
```

Output

```
Request:method=putServData ...
--------------------------
OK – putServData
--------------------------
```

delEntryData

Use

This method deletes an index/queue from the name server databasis.

Only use this method if errors have occurred and if you are certain that you need to delete the entry. Deleting an entry can interfere with TREX functionality. Check your entries again before executing this method.

Features

Entry

Server Group	Optional For this method, only `indexserver` or `queueserver` are sensible.
Address	Optional Address of the server. For the format of the address, see showAllMembers [Page 117].
Entry	Entry that you want to delete. Enter the index or queue in the format `index_<index_ID>.`

Output

The output area displays your entries again. If the entry was deleted, you receive a confirmation. If not, you receive a message telling you that the specified entry does not exist.

Example

Deleting an Index Entry

Entry

Server Group: **`indexserver`**

Address: **`mytrexhost2:8351`**

Entry: **`index_marketing`**

Output

```
Request:method=delEntryData ...
-------------------------
OK - Entry deleted!
-------------------------
```

setInactive

Use

This method marks a server as inactive.

During routine operation, this indicator is set automatically. Before a server stops, it registers its intention to stop with the name server and is then marked as inactive. The server and any indexes or queues stored on it are still recognized system-wide, but are not available for use.

In the following cases you have to mark a server as inactive manually.

- For test purposes, you temporarily do not want to use a running server.
- Only if the watchdog is not activated: A server is not available due, for example, to a power cut. It can therefore not register itself as inactive at the name server.

If the watchdog is activated, an unavailable server is automatically marked as inactive. For more information on the watchdog, see Activating the Watchdog [Page 43]

Features

Input

Field	Description
Address	Address of the server that you want to mark as inactive. For the format of the address, see showAllMembers [Page 117].

Output

A message tells you that the server is now marked as inactive.

Example

Marking the index server as inactive

Input

Address: **`mytrexhost2:8351`**

Output

```
Request:method=setInactive …
--------------------------
OK – setInactive
--------------------------
```

setActive

Use

This method marks a server as active. You use this method if

- You want to register another name server and thereby activate name server replication.
- You have previously marked a server as inactive for test purposes. You now want to bring the server back into operation, and mark it as active again.

Features

Input

Field	Description
Server Group	Group of the server that you want to mark as active; `indexserver`, `queueserver`, `httpserver`, `nameserver`, or `preprocessor`
Address	Address of the server. For the format of the address, see showAllMembers [Page 117].

Output

A message tells you that the server is now marked as active.

Example

Register a name server and activating name server replication

Input

Server Group: **`indexserver`**

Address: **`mytrexhost2:8355`**

Output

```
Request:method=setActive ...
--------------------------
OK – setActive, created
--------------------------
```

Mark the index server as active

Input

Server Group: **`indexserver`**

Address: **`mytrexhost2:8351`**

Output

```
Request:method=setActive ...
--------------------------
OK – setActive
--------------------------
```

setAsDefault

Use

This method marks a server as the default server.

This is only relevant for index replication. It controls the index servers on which new indexes are to be created. If you want original indexes to be located on the masters and not on the slaves, the masters have to be marked as default servers.

The name server distributes requests for the creation of an index as follows:

- If index servers are marked as default, the name server only considers the servers in question for distribution.
- If no index servers are marked as default, the name server considers all index servers for distribution.

Features

Input

Field	Description
Server Group	Group of the server that you want to mark as the default server. For this method, only `indexserver is` sensible.
Address	Address of the server in the format `<hostname>:<port>`.

Output

A message tells you that the server is now marked as default.

Example

Indicate that the master is the default index server for index replication

Input

Server Group: **`indexserver`**

Address: **`mytrexhost1:8351`**

Output

```
Request:method=setAsDefault ...
--------------------------
OK – setAsDefault
--------------------------
```

reloadData

Use

This method causes the name server to reload its databasis from the hard drive.

You use this method if you have changed the name server databasis manually and want to activate the changes without restarting the name server.

Features

Input

No input required.

Output

A message tells you that the databasis was loaded.

saveData

Use

This method causes the name server to save its databasis on the hard drive.

Only use this method if errors have occurred. During normal operation, the name server automatically saves its databases to the hard drive as soon as the databasis is changed.

Features

Input

No input required.

Output

A message tells you that the databasis was saved.

Example

You determine that the name server does not have write-access to the directory in which its databasis is located (`<TREX_Directory>\name_server`). After changing the permissions for the directory, you can save the databasis on the hard drive by choosing *saveData.* You can check that the permission problem has been solved at the same time.

watchDog

Use

This method changes the status of the watchdog or returns its current status. For more information on the watchdog, see Activating the Watchdog [Page 43].

Features

Input

Field	Description
Action	Action that you want to carry out on the watchdog. Possible entries are: • **`start`** • **`stop`** • (no entry) – return current status

Output

A message displays the current status or the status change.

getAddressListforEntry

Use

This method displays the addresses of the servers on which an index or queue is located.

You can also use this method to display the addresses of a group of server, for example, of all name servers.

Features

Entry

Server Group	Group of servers whose addresses you want to display; `indexserver`, `queueserver`, `httpserver`, `nameserver`, or `preprocessor`
Entry	Optional Entry for which you want to display the server addresses. Make the entry in the format `index_<index_ID>`.

Output

The output area displays the addresses in the format `<hostname>:<port>`.

Example

On which index server is the index ‘marketing’ located?

Input

Server Group: **indexserver**

Entry: **index_marketing**

Output

```
Request:method=getAddressListforEntry ...
-------------------------
OK – Data follows:
mytrexhost1:8351
mytrexhost2:8351
mytrexhost3:8351
-------------------------
```

What are the addresses of the name servers?

Input

Server Group: **nameserver**

Entry:

Output

```
Request:method=getAddressListforEntry ...
-------------------------
OK – Data follows:
mytrexhost1:8355
mytrexhost2:8355
-------------------------
```

getEntryInfo

Use

This method displays the parameters of an index or queue in the name server databasis.

Features

Input

Field	Description
Entry	Entry whose parameters you want to display. Make the entry in the format `index_<index_ID>`.
Address	Address of the index or queue server to which the entry belongs. Enter the address in the format `<hostname>:<port>`.

Output

The output area displays the parameters of the entry. For more information on the meaning of parameters, see Name Server Databasis [Page 134], section [index_<index_ID>].

Example

What parameter does the index ‘marketing’ have in the name server databasis?

Input

Entry: **`index_marketing`**

Address: **`mytrexhost1:8351`**

Output

```
Request:method=getEntryInfo ...
--------------------------
OK – Data follows:
is_searchable=true
is_writable=true
```

What parameter does the queue for the index ‘marketing’ have in the name server databasis?

Input

Entry: **`index_marketing`**

Address: **`mytrexhost1:8352`**

Output

```
Request:method=getEntryInfo ...
--------------------------
OK – Data follows:
queue_available=true
```

deleteService

Use

This method deletes a server from the name server databasis.

Only use this method if errors have occurred and if you are certain that you need to delete the server data. Deleting server data can interfere with TREX functionality. Check your entries again before executing this method.

Features

Entry

Field	Description
Address	Address of the server whose data you want to delete. For the format of the address, see showAllMembers [Page 117].

Output

The output area tells you that the data was been deleted.

Example

You implemented a TREX system with index replication and want to permanently remove a slave from the group of servers. The name server databasis should no longer receive data on this slave.

Input

Address: **`mytrexhost2:8351`**

Output

```
Request:method= deleteService ...
-------------------------
OK - Service deleted
```

Test Replicated Name Servers

Use

This method tests whether the replication of the name server databasis is functioning. The system carries out the following steps during the test:

- Checks that the name server can be reached.
- Adds test data to the databases of all name server.
- Checks after a certain amount of time that the text data is available in all databases.

Prerequisites

- You have carried out all steps necessary for name server replication (see Name Server Replication [Page 48]).
- The name servers are running.

Features

Entry

You can call up this method using the *Advanced* menu and specify the addresses of all name servers that are to participate in replication.

Address format: `<hostname>:<port>` The standard port is 8355.

Output

The output area displays the steps that are carried out during the test. A report is also displayed, telling you whether the test was successful.

The following error messages may appear:

Errors when testing replication

Error	Cause	Troubleshooting
`Host <hostname1>:<port1> is not connected to <hostname2>:<port2>`	The name servers are not recognizing one another.	Activate the additional name server (see Activating the New Name Server [Page 52]). This registers the new name server with the other name server, and means that replication of the databasis can now take place.
`Service on <hostname>:<port> is not running`	The name server is not running.	Check whether the TREX daemon is running on the specified host. • If it is not running, start it. • If it is running, check that the name server is entered in the configuration of the TREX daemon (see Configuring the TREX Daemon [Page 50]).

Auto Repair Replicated Name Servers

Use

This method repairs inconsistencies in the name server databases.

Prerequisites

The name servers are running.

Features

Input

You can call up this method using the *Advanced* menu and specify the addresses of all name servers that are participating in replication.

Address format: `<hostname>:<port>` The standard port is 8355.

Output

The output area displays the steps that are carried out when the name server databases are synchronized. A report is also displayed, telling you whether the synchronization was successful.

The following error message may appear:

Error when synchronizing name server databases

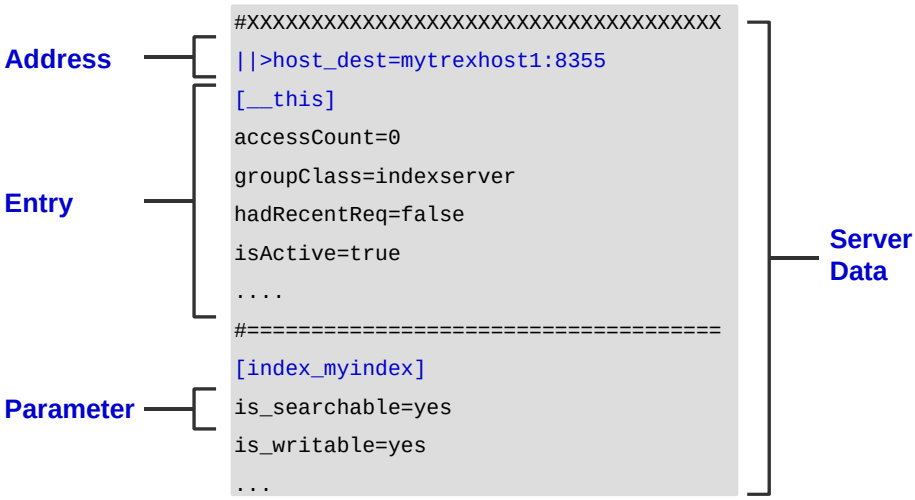
Error	Cause	Troubleshooting
`Host <hostname1>:<port1> is not connected to <hostname2>:<port2>`	The name servers are not recognizing one another.	Activate the additional name server (see Activating the New Name Server [Page 52]). This registers the new name server with the other name server, and means that replication of the databasis can now take place.

Name Server Databasis

The name server databasis contains information on all TREX servers known to the name server.

Structure

The data for a server is structured as follows:



Description

Address

`host_dest=<address>`

Address of the server in the form

- `__me__`, if the data relates to the name server whose databasis you are displaying
- **`http://<hostname>:<port>/trex`** (UNIX) or **`http://<hostname>:<port>/trexhttpserver/trexhttpserver.dll`** (Windows) if the data relates to a Web server
- `<hostname>:<port>`, if the data relates to another server

[__this]

This entry contains general information on the server.

accessCount=<number>

Specifies how often the name server can forward the address of the server to a querying component. The name server consults this count for load distribution. If the name server receives a request that could be answered by more than one server, it returns the server with the lowest count.

You are using two Web servers for TREX.

The address of Web server 1 has been forwarded 5000 times.

The address of Web server 2 has been forwarded 4990 times.

Content Management requests the address of a Web server in order to forward a search query. The name server forwards the address of Web server 2 and raises the count for this server.

The count does not return how many times the server was actually accessed. It only states the number of times the address of the server was forwarded. The number of actual accesses is normally higher. This is because the TREX servers and Content Management store the data received from the name server and communicate using the stored data for a certain amount of time. Therefore, the address of a server is not always queried at the name server before it is accessed.

groupClass=<group>

Group to which the server belongs: `httpserver`, `indexserver`, `nameserver`, `queueserver`, or `preprocessor`.

hadRecentReq=<true|false>

Not relevant.

isActive=<true|false>

Specifies whether the server is available.

IsDefault=<true|false>

Specifies whether the server is marked as the default server.

Default value: `false`

This parameter is only relevant for index replication. It controls the index servers on which new indexes are to be created. Normally, original indexes should be created only on masters, not on slaves. To achieve this, the master server has to be marked as the default server.

The name server distributes requests for the creation of an index as follows:

- If index servers are marked as default, the name server only considers the servers in question for distribution.
- If no index servers are marked as default, the name server considers all index servers for distribution.

lastPublished=<date>

Specifies when the server last published new information.

locationGroup=<grouping>

Not relevant.

[_config]

Only for name servers.

This entry contains specifications on the configuration of the name server. Only the parameter `usereplication` is currently relevant. The other parameters are not considered.

`usereplication=<true|false>`

Specifies whether name server replication is active. This parameter is automatically set to `true` when you activate an additional name server.

Default value: `false`

[_watchdog]

Only for name servers.

This entry contains specifications on the configuration of the watchdog.

`interval=<seconds>`

Interval (in seconds) at which the watchdog checks that the server can be reached.

Default value: 10 seconds

`isActive=<true|false>`

Specifies whether the watchdog is activated permanently.

Default value: `false`

[index_<index_ID>]

Only for index and queue servers.

This entry contains information on an index or queue.

Information on an Index

`is_searchable=<yes|no>`

Specifies whether read-access if available for the index (whether the index can be used for search queries).

Default value: `yes`

`is_writable=<yes|no>`

Specifies whether write-access is available for the index.

Default value: `yes`

If an index is replicated, the parameter has the value `yes` on the master and the value `no` on the slave.

Information on a Queue

`queue_available=true`

Specifies whether the queue is available. This parameters exists for all queues but is not currently relevant. This parameter always has the value `true`.

[process_<preprocessor_process>]

Only for a preprocessor.

Each preprocessor tells the name server databasis which processes it can carry out and the sequence in which it can do so.

These specifications are not currently relevant.