DEVELOPER'S GUIDE

IMPORTANT NOTICE

This documentation is provided without a fee. The content of this documentation is provided for informational use only, is subject to change without notice, may contain technical inaccuracies or typographical errors, and should not be construed as a commitment of Lursoft IT. Lursoft IT may make improvements or changes in the product described in this documentation at any time without notice.

COPYRIGHT

CONTACT INFORMATION

Email: support@siets.net

Website: www.siets.net

TABLE OF CONTENTS

PREFACE....6

Audience....6
Structure of This Guide....6
Related Information....7
Typographic Conventions....7
Abbreviations....7

1. INTRODUCING SIETS....8

1.1. What is SIETS?....8
1.2. SIETS in Corporate Networks....9
1.3. Understanding SIETS Environment....10
1.3.1. Overview....10
1.3.2. Accessing SIETS Server....11
1.4. Concepts....12
1.4.1. SIETS Server....12
1.4.2. SIETS FTS Capability....12
1.4.3. SIETS API....12
1.4.4. SIETS Web Server Module....13
1.4.5. SIETS Console....13
1.4.6. SIETS Document....13
1.4.7. SIETS Storage....13
1.4.8. Vocabulary....13
1.4.9. Document Repository....13
1.4.10. Inverted Index....13
1.5. SIETS Architecture....14
1.5.1. Client — Server Architecture....14
1.5.2. Multiple Storages Architecture....14
1.5.3. Multi-Server Architecture....15
1.5.4. Understanding Storing Information in SIETS....16
1.5.5. Indexing Documents in SIETS Storage....17
1.5.6. Querying SIETS Storage....18
1.6. Standards Compatibility....19
1.7. Features....20
1.8. SIETS and the Future....20

2. UNDERSTANDING SIETS DOCUMENT STRUCTURE....21

2.1. Overview....21
2.2. Creating Document Structure with Application....21
2.3. Importing XML Structured Data....22
2.4. Document Ordering in Result Set....24
2.4.1. Overview....24
2.4.2. Rate....26
2.4.3. Relevance....26

3. INTERNATIONALIZATION....30

3.1. Multi-language Support and Character Encoding....30
3.1.1. Overview....30
3.1.2. Storing and Searching in Single Encoding....32
3.1.3. Storing in Different Encodings and Searching in Multiple Bytes per Character Encoding....33
3.1.4. Storing in Different Encodings and Searching in One Byte per Character Encoding....34

3.2. Formatting XML Special Characters..... 35

4. SIETS API SPECIFICATION..... 36

4.1. Overview..... 36
4.1.1. Submitting SIETS Commands and Receiving Replies..... 36
4.1.2. XML Message Structure..... 38
4.2. SIETS XML Message Envelope..... 39
4.3. Data Manipulation..... 40
4.3.1. Insert, Update, and Replace..... 40
4.3.2. Delete..... 42
4.3.3. Index..... 42
4.3.4. Clear..... 42
4.3.5. Get_scheme..... 43
4.3.6. Set_scheme..... 43
4.4. Status Monitoring..... 44
4.4.1. Status..... 44
4.5. Data Retrieval..... 46
4.5.1. Lookup and Retrieve..... 46
4.5.2. Search..... 47
4.5.3. Select..... 58
4.5.4. Similar..... 59
4.5.5. Alternatives..... 60
4.5.6. List-last..... 61
4.6. Alerts..... 62
4.6.1. Adding trigger..... 62
4.6.2. Removing trigger..... 62
4.6.3. Clear triggers..... 62
4.6.4. Examining document against triggers..... 62
4.6.5. Configuration parameters..... 62
4.7. Error Handling..... 63

5. SIETS CLUSTERING..... 64

5.1. Principles..... 64
5.2. Creating SIETS Cluster..... 64

6. SIETS CONSOLE..... 65

6.1. Overview..... 65
6.2. Parameters..... 65
6.3. SIETS Console Parameters..... 65
6.4. Templates..... 66
6.5. Built-in Commands..... 66
6.6. Running SIETS Console..... 67

7. MESSAGE FILES..... 68

7.1. Overview..... 68
7.2. Structure of Message Files..... 68
7.3. Running siets-load..... 69

8. USE CASES..... 70

8.1. Use Case in C: Importing Text Files..... 70
8.2. Use Case in Perl: Importing Text Files..... 74
8.3. Use Case in PHP: Searching SIETS Storage and Returning Results in HTML..... 77
8.4. Use Case in ASP: Searching SIETS Storage and Returning Results in HTML..... 82

8.5. Use Case in Java: Searching SIETS Storage from applet....84
8.5.1. SietsJApi.java....84
8.5.2. SietsMess.java....86
8.5.3. SietsExch.java....87
8.5.4. SietsXMLParser.java....89
8.6. Use Case in FoxPro I: Updating SIETS Storage from Database Table Data....90
8.7. Use Case in FoxPro II: Searching SIETS Storage and Returning Results to Cursor....93
APPENDIX A: ERROR MESSAGES....96
Reporting Problems....97
APPENDIX B: API REFERENCE FOR FOXPRO....98
Microsoft Visual FoxPro....98
APPENDIX C: FREQUENTLY ASKED QUESTIONS....99
GLOSSARY....102
INDEX....104

PREFACE

This preface is an introduction to the SIETS Developer's Guide. It defines the audience, describes the structure of this guide, and lists typographic conventions and abbreviations used throughout the guide.

This guide is compliant with the SIETS server version 3.3.19.

This section contains the following topics:

- Audience
- Structure of This Guide
- Related Information
- Typographic Conventions
- Abbreviations

Audience

This guide is intended for application developers. In SIETS context these are developers that build customized applications based on SIETS API.

Structure of This Guide

This guide has the following structure:

Section	Description
Introducing SIETS	Describes SIETS, its concepts, and architecture.
Understanding SIETS Document Structure	Describes SIETS document structure and presents scenarios for unstructured source data and XML structured source data.
Internationalization	Describes multi-language support and text encoding concepts, and explains XML formatting concepts.
SIETS API Specification	Contains all SIETS function descriptions and syntax in XML.
SIETS Clustering	Describes SIETS clustering.
SIETS Console	Explains how to run and work with SIETS console.
Message Files	Describes the concept of message files and explains how to run the siets-load command.
Use Cases	Lists and describes sample applications that are based on SIETS API commands.
Appendix A: Error Messages	Contains a list of error messages.
Appendix B: API Reference for FoxPro	Contains SIETS API reference for Microsoft Visual FoxPro programming language.
Appendix C: Frequently Asked Questions	Contains a list of frequently asked questions and answers to them.

Related Information

The SIETS package includes the following guides:

Title	Description
SIETS Administration and Configuration Guide	Describes the SIETS administration and configuration concepts and contains step-by-step instructions.
SIETS Installation Guide	Describes how to install SIETS.

Typographic Conventions

The following styles and conventions are used in this guide:

Convention	Description
Verdana	Represents command, function, file and directory names, system messages, and command-line commands.
Hyperlink	Represents a hyperlink. Clicking on this field takes you to the identified place.
Example	Represents an example.
Source code	Represents code.
Comment	Represents a comment in the code.

Abbreviations

The following abbreviations are used in this guide.

Abbreviation	Description
API	Application programming interface.
FTS	Full text search.
XML	Extensible markup language.
SQL	Structured query language.
UTF	UCS (universal character set) transformation format.
HTTP	Hypertext transport protocol.

1. INTRODUCING SIETS

This guide introduces SIETS from an application developer's perspective and provides reference material for building customized applications based on SIETS.

This section includes the following:

- What is SIETS?
- SIETS in Corporate Networks
- Understanding SIETS Environment
- Concepts
- SIETS Architecture
- Standards Compatibility
- Features
- SIETS and the Future

1.1.What is SIETS?

SIETS is a system for information storage and retrieval. The SIETS system consists of the SIETS server and application programming interface (API) for building information storage and retrieval applications.

The SIETS server is an operational unit that performs information storing and retrieval tasks by executing a predefined set of commands.

SIETS API is used for building applications that are specific and customized according to your company needs.

Note: With the SIETS package integrated applications are delivered for most common source data formats and retrieval scenarios. However, it is not possible to cover all possible scenarios, therefore, it is within the scope of this guide to provide you reference material for building your own applications.

Nowadays, unstructured data amounts in companies are increasing very rapidly; the only way how to effectively retrieve such data from collections and, therefore, make the data usable, is full text search (FTS). Full text search is the main methodology implemented in SIETS server for information indexing and searching.

Full text search in SIETS is based on an optimized mathematical model, which ensures very high performance for searching poorly structured information in large amounts compared to traditional SQL systems. For this purpose, in SIETS, data are stored in an inverted index.

For more information on how data are stored in SIETS, see Understanding Storing Information in SIETS.

Subjects for full text search can be any unstructured data, for example, text collections, separate phrases or words in text documents, Web pages, Web addresses, several special markups for textual and numerical data, bookmarks of HTML or XML pages, domain names, SQL database entry key IDs, file names, and so on.

The following figure illustrates the SIETS system from a high level:

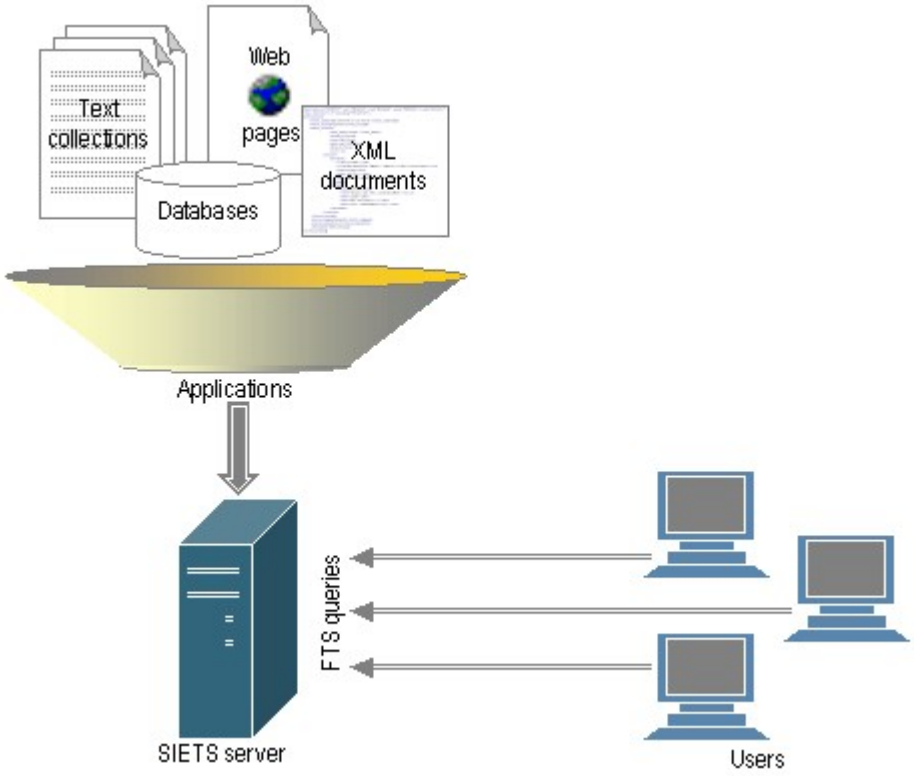

Figure 1: SIETS operational diagram

In Figure 1, users are accessing the SIETS server via FTS queries. However, also other technologies for data storing, manipulating and other implemented in SIETS, for example, retrieval queries, update requests, status and control commands, XML queries using XPath notation, and so on.

1.2.SIETS in Corporate Networks

SIETS system can be integrated into an existing corporate network system. The SIETS server is incorporated into the network system just like any other server. The following figure describes a sample corporate network with the SIETS server:

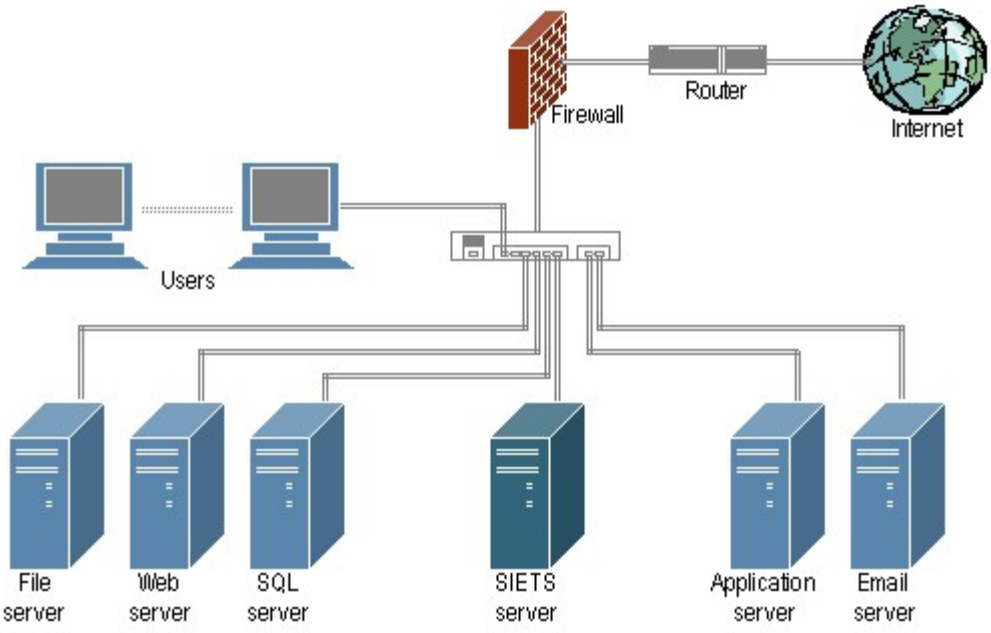

Figure 2: SIETS server in a corporate network

Application servers and transaction processors from an existing corporate network can access the SIETS server as active SIETS API clients; in that case, for security reasons, end users cannot directly access the SIETS server. In that way, SIETS server can be used in any corporate network, independently from the operation system, database environment, or programming language used for application development.

For very large data amounts, SIETS supports sever clustering, which implies performance scalability.

For more information on SIETS server multi-server architecture, see Multi-Server Architecture.

1.3.Understanding SIETS Environment

This section describes SIETS environment from users perspective.

This section contains the following topics:

- Overview
- Accessing SIETS Server

1.3.1.Overview

As already described in the previous section, the SIETS system is used for data storage and retrieval. The SIETS server is an operational unit in the SIETS system that performs these tasks. There is a predefined set of commands that are understood and executed by the SIETS server. The commands are implemented as XML requests and replies and transported via HTTP POST.

This implies that for sending commands to the SIETS server, first these commands must be created in the form of XML request messages. However, to make sending commands to the

SIETS server easier, a server side mechanism that reads HTTP GET parameters and composes XML request messages from the parameters is included in the SIETS system. In that case you do not have to worry about creating XML request messages, but only have to pass right parameters, from which XML request messages are automatically created and sent to the SIETS server.

Of course, you can also create XML messages at the application side at your own convenience.

In a similar way, there is also a mechanism that formats XML reply messages received from the SIETS server by using an XSLT stylesheet. Again, it is your decision whether to handle XML reply messages on the application side, or to create an XSLT stylesheet using which results received from the SIETS server are automatically formatted and can be directly passed to end users.

1.3.2.Accessing SIETS Server

The following diagram describes how the SIETS server is accessed:

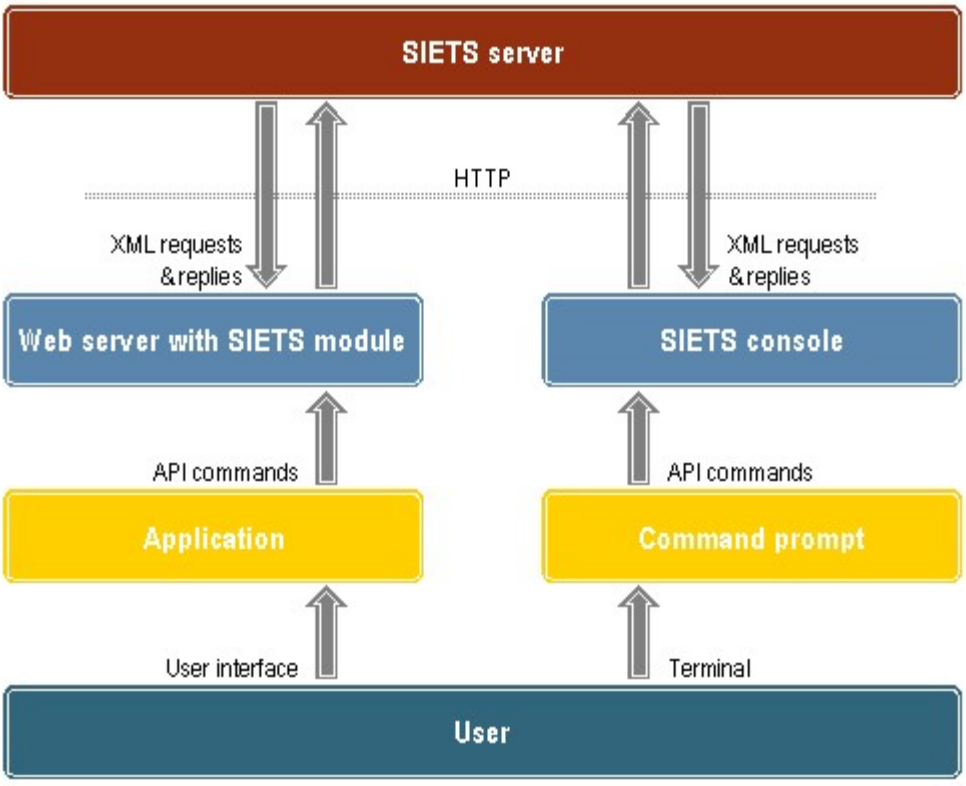

Figure 3: Accessing SIETS server

The following steps provide a general description of how the SIETS server is accessed:

1. Users enter commands, such as search queries, for the SIETS server from a user interface of an application, for example, a Web search form.
2. A custom built application calls a SIETS API command with its parameters.
3. The Web server receives HTTP request and passes it to the SIETS Web server module using Comman Gateway Interface (CGI) of Apache API.

4. The SIETS Web server module translates each user command into an XML request and submits it to the SIETS server via UNIX domain sockets.
5. The SIETS server responds to the application returning XML replies that are optionally formatted using XSLT stylesheet and then can be displayed and viewed through the application user interface, for example, a Web page.

The steps just describe implies that the SIETS server is accessed using SIETS API.

However, for debugging purposes the SIETS server also can be accessed via SIETS console.

For a detailed SIETS console description, see SIETS Console.

1.4.Concepts

This section introduces and briefly explains concepts that readers must be familiar with before going into details.

This section contains the following topics:

- SIETS Server
- SIETS FTS Capability
- SIETS API
- SIETS Web Server Module
- SIETS Console
- SIETS Document
- SIETS Storage
- Vocabulary
- Document Repository
- Inverted Index

1.4.1.SIETS Server

SIETS server is a stand-alone server for storing and retrieving information such as plain texts or XML structured documents. It can be run in one or more instances per computer.

For more information, see Multiple Storages Architecture.

1.4.2.SIETS FTS Capability

SIETS is designed to support retrieving information stored using full text search queries.

1.4.3.SIETS API

SIETS application programming interface (API) is a standardized set of commands for accessing the SIETS server.

1.4.4.SIETS Web Server Module

SIETS Web server module is a module integrated with the Web server that receives requests from an application through the Web server via HTTP POST and dispatches them to the SIETS server via UNIX domain sockets.

Also, functionality of composing XML request messages from HTTP GET or POST parameters and optional formatting of the XML reply messages with a given XSLT stylesheet is included the SIETS Web server module.

The SIETS system is designed so that the SIETS server module can be integrated with the Web server through the Common Gateway Interface (CGI) or Apache API. Thus, it can be integrated with any Web server through CGI, and also it can be integrated with the Apache Web server through Apache API, which increase effectives of the whole system.

1.4.5.SIETS Console

SIETS console is a simple text application for accessing the SIETS server directly using the same functions as in SIETS API.

1.4.6.SIETS Document

SIETS document is a unit in the SIETS storage against which searching is performed. It can be unstructured or XML structured.

1.4.7.SIETS Storage

SIETS storage is a data collection for storing SIETS documents in a format that ensures a search is performed very fast. The SIETS storage is serviced by one SIETS server instance, and consists of vocabulary, document repository, and inverted index. Multiple storages can be run on a single computer.

1.4.8.Vocabulary

Vocabulary is a list of all unique words in the SIETS storage. Unique words are found in documents and added to the vocabulary while storing these documents to the SIETS storage. Each SIETS storage has its own vocabulary. Each word in the vocabulary has an ID of the integer type assigned to it. Vocabulary is stored in RAM for better performance.

1.4.9.Document Repository

Document repository is a place where all SIETS documents are stored in the format, in which they were stored in the SIETS system, for returning the documents on a search request. Each SIETS storage has its own document repository.

1.4.10.Inverted Index

Inverted index is a list of words, where each word has a list of pointers to SIETS documents in which the word occurs. Inverted index ensures fast FTS functionality with possibility to build different logical expressions when performing a search. Each SIETS storage has its own inverted index.

1.5.SIETS Architecture

This section describes SIETS from various architectural perspectives.

This section contains the following topics:

- Client — Server Architecture
- Multiple Storages Architecture
- Multi-Server Architecture
- Understanding Storing Information in SIETS
- Indexing Documents in SIETS Storage
- Querying SIETS Storage

1.5.1.Client — Server Architecture

The following figure describes SIETS architecture from the client — server perspective:

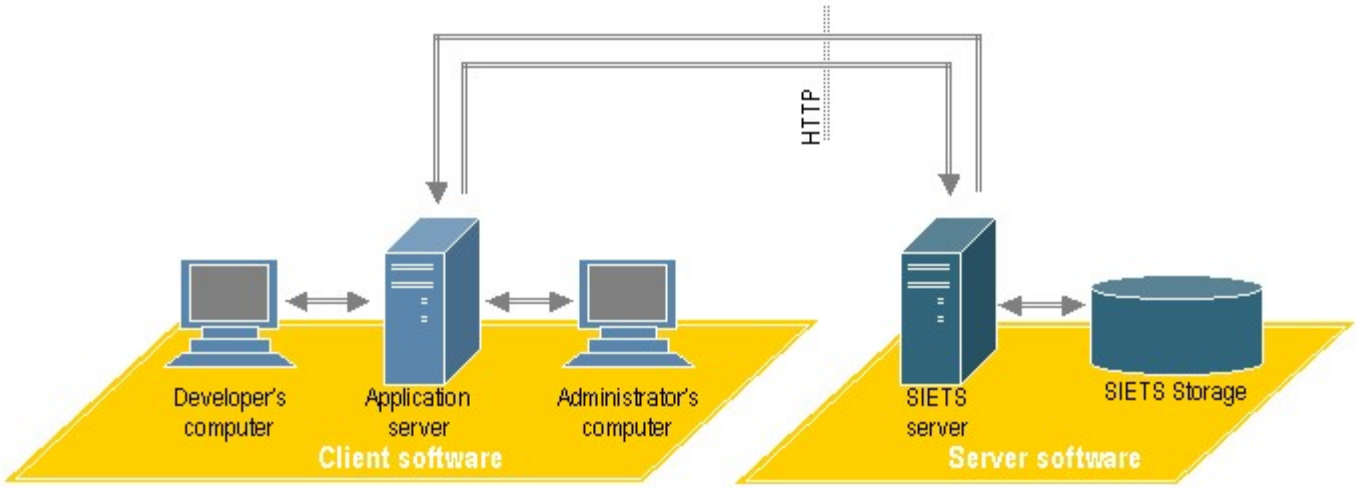

Figure 4: SIETS client — server architecture

From the client — server perspective, the SIETS system consists of the client part and the server part.

On the client side, developers and administrators work with an application server, which initializes SIETS API command calls and sends them to the SIETS server via HTTP.

On the server side, the SIETS server executes these SIETS API commands accessing data in the SIETS storage and sends a reply back to the client side's application server.

1.5.2.Multiple Storages Architecture

The following figure describes how multiple SIETS server instances can be run on a single SIETS server:

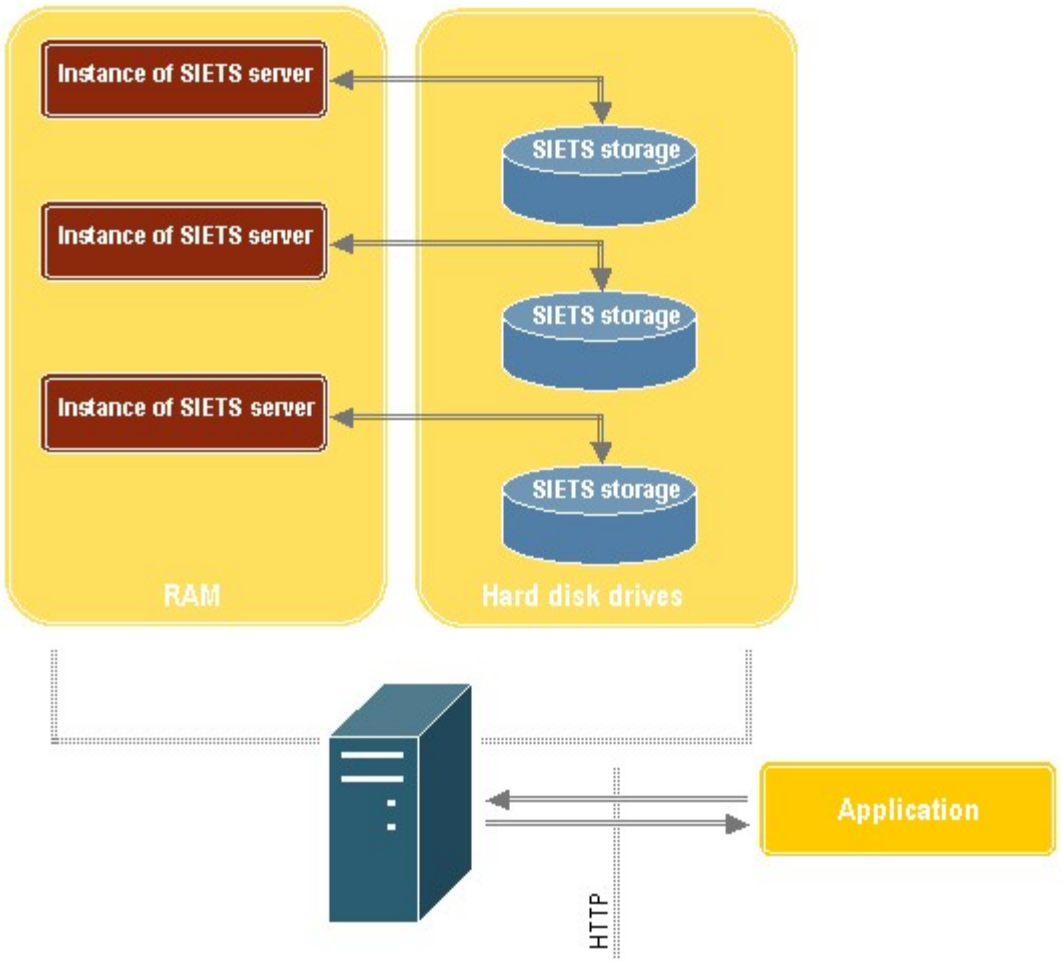

Figure 5: SIETS multiple storages architecture

Multiple instances of the SIETS server can be run on a single computer, which each works with its own SIETS storage.

1.5.3.Multi-Server Architecture

To ensure scalability of larger amounts of data, the SIETS server can be clustered sharing a single SIETS storage across many computers.

The following figure describes how a single SIETS storage can be distributed on several SIETS servers:

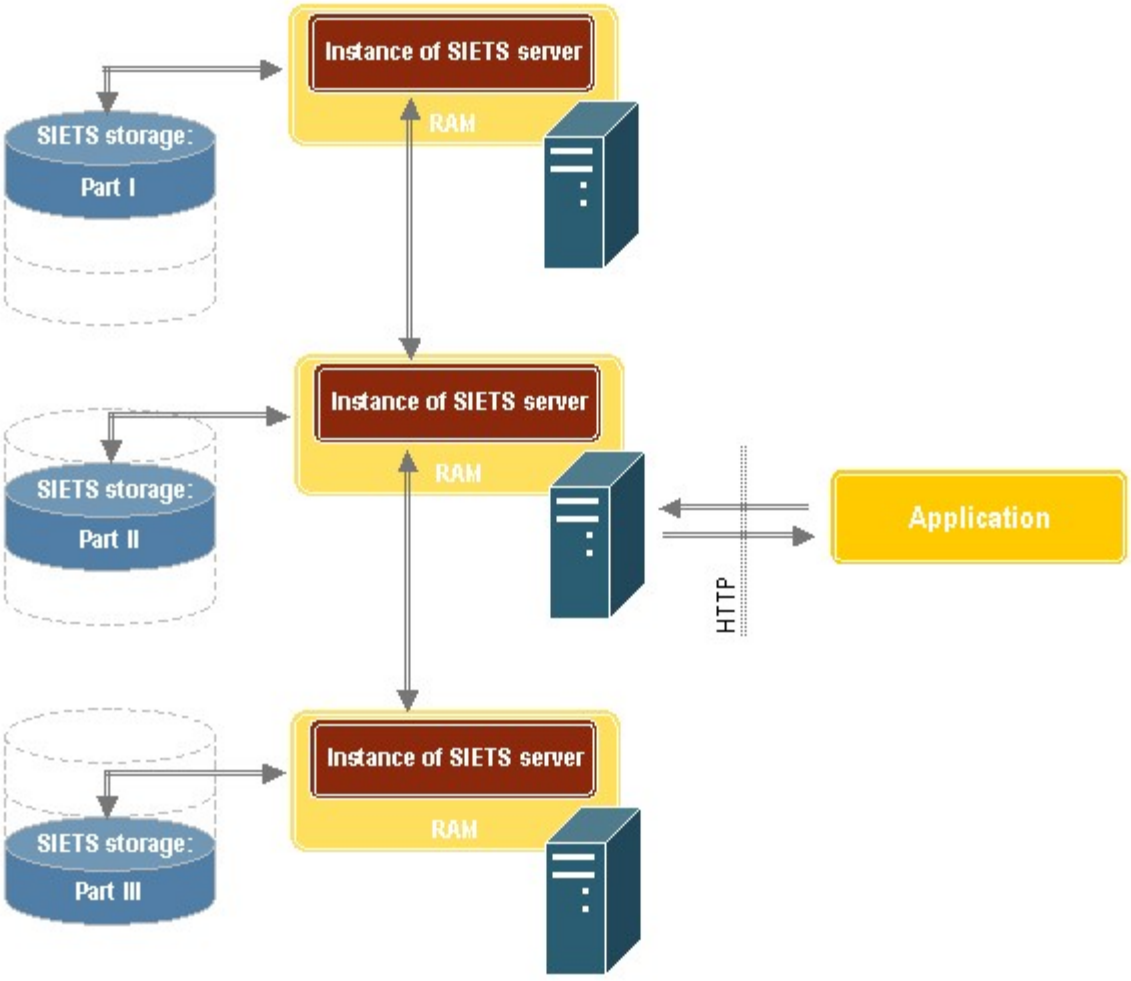

Figure 6: SIETS multi-server architecture

For more information on SIETS clustering, see SIETS Clustering.

1.5.4.Understanding Storing Information in SIETS

The following figure describes how data are imported and stored in SIETS:

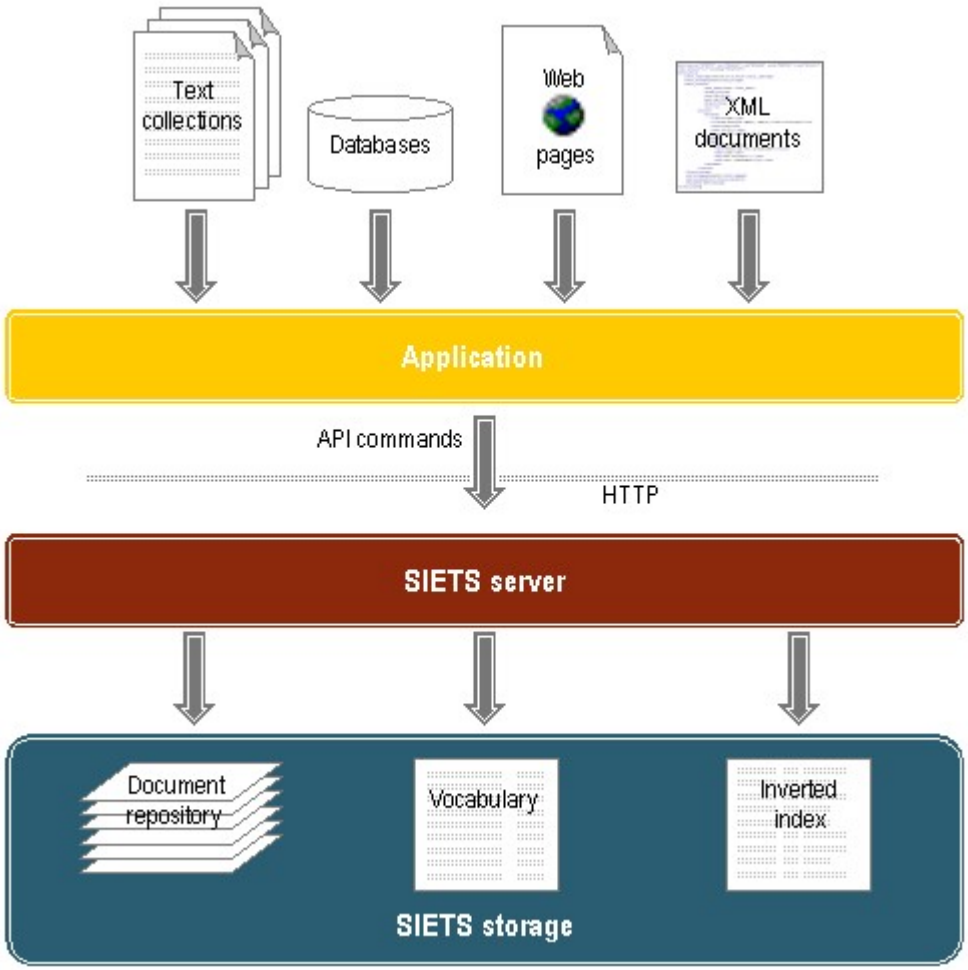

Figure 7: Storing information in SIETS

1. Data are entered by end users in custom built applications.
2. Using SIETS API commands data are submitted to the SIETS server via HTTP.
3. From the submitted data, the SIETS server creates an inverted index, vocabulary, and document repository, which all are contained by the SIETS storage.

For more information on the SIETS storage, see Indexing Documents in SIETS Storage and Querying SIETS Storage.

1.5.5.Indexing Documents in SIETS Storage

Note: This section contains some of SIETS system implementation details. Description provided in this section is very general and does not include implementation details for all SIETS functionality.

Note: The knowledge provided in this section is not required for SIETS application developers. However, it can be found useful for a better understanding of the SIETS system.

The following figure describes general principles how a document is indexed in the SIETS storage:

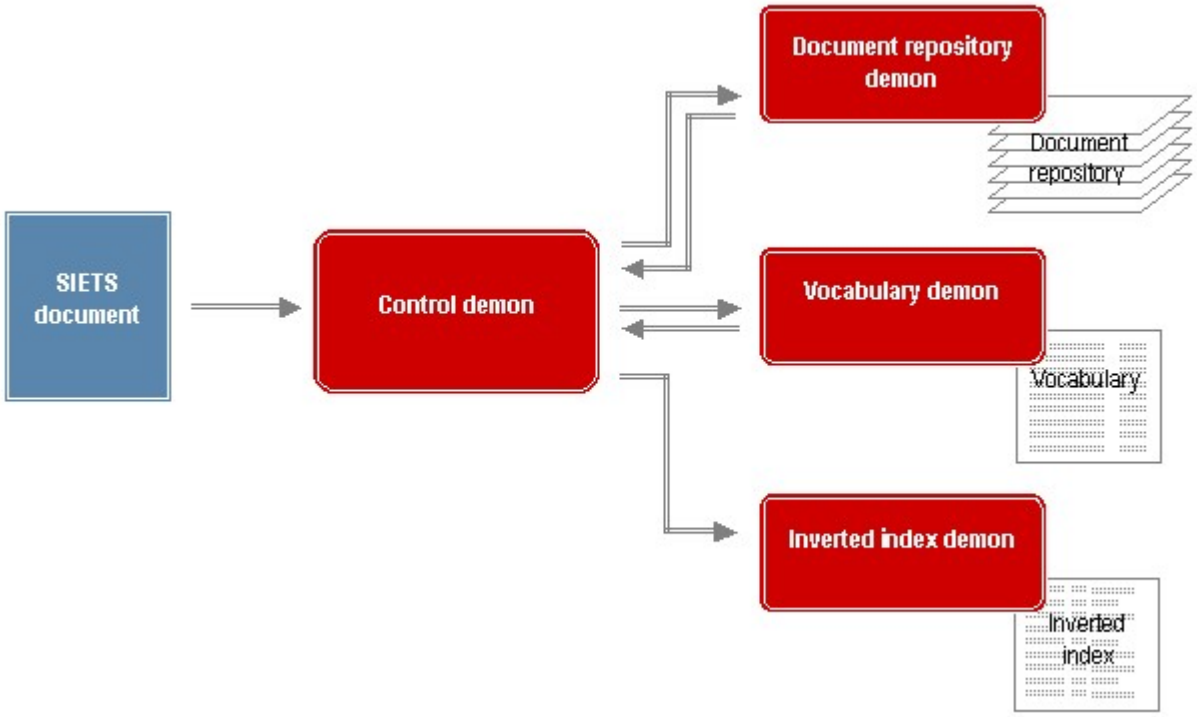

Figure 8: Indexing documents in SIETS storage

1. When the SIETS server receives an XML request containing document that must be imported in the SIETS storage, the Control demon[1] parses the XML request.
2. The Control demon sends the document to the Document repository demon.
3. The Document repository demon stores the document in the document repository, assigns a unique ID of the integer type to it and sends the ID to the Control demon.
4. The Control demon sends all textual data from the document to the Vocabulary demon.
5. The Vocabulary demon translates all words in the document to unique IDs of the integer type and sends them to the Control demon.
6. The Control demon sends the document ID and all IDs of the words contained by it to the Inverted index demon.
7. The inverted index demon links word IDs with the document ID and inserts them in the inverted index.

1.5.6.Querying SIETS Storage

Note: This section contains some of SIETS system implementation details. Description provided in this section is very general and does not include implementation details for all SIETS functionality.

Note: The knowledge provided in this section is not required for SIETS application developers. However, it can be found useful for a better understanding of the SIETS system.

The following figure describes general principles how a query is processed in the SIETS storage:

[1] A demon is a program or process, part of a larger program or process, that is dormant until a certain condition occurs and then is initiated to do its processing.

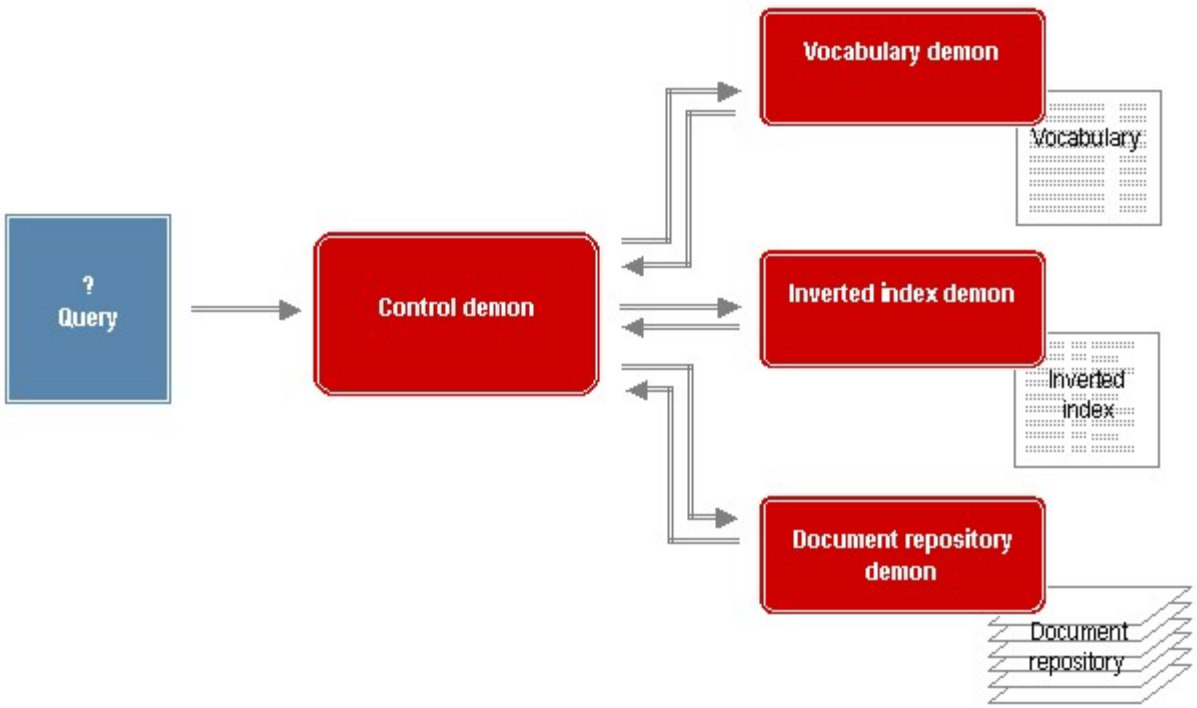

Figure 9: Querying SIETS storage

1. When the SIETS server receives an XML request containing a query, the Control demon parses the XML request.
2. The Control demon sends the query to the Vocabulary demon.
3. The Vocabulary demon translates words from the query to IDs and sends them to the Control demon.
4. The Control demon sends the IDs to the Inverted index demon.
5. The Inverted index demon searches and returns from the inverted index to the Control demon a list of document IDs, which are linked to the query word IDs.
6. The Control demon sends the list of document IDs to the Document repository demon.
7. The Document repository demon searches and returns a result set containing a document list that matches the query.

1.6.Standards Compatibility

SIETS is designed to comply with the following standards:

Standard	Reference
XML 1.0	http://www.w3.org/TR/REC-xml
UTF-8	RFC 2279: UTF-8, a transformation format of ISO 10646
HTTP	Hypertext Transfer Protocol
XPath 1.0	http://www.w3.org/TR/xpath

1.7.Features

The SIETS features are listed and referred to a section in this guide, in which it is described, in the following table:

Title	Section
FTS	Search
Relevance	Relevance
Multi-language support	Multi-language Support and Character Encoding
Case support	Search
Boolean expressions	Boolean Expressions
Stemming	Stemming
Wildcard search	Wildcard Patterns
Fuzzy search	Alternatives
Markup search	Search within Markup

1.8.SIETS and the Future

It is planned for the nearest SIETS future releases:

- to support public key cryptography for document encryption and authentication

2. UNDERSTANDING SIETS DOCUMENT STRUCTURE

This section describes SIETS document structure concepts and explains strategies, if source data that you want to import into the SIETS system, are unstructured, and if the source data are XML structured. It also describes document ordering and language and text encoding concepts.

This section contains the following topics:

- Overview
- Creating Document Structure with Application
- Importing XML Structured Data
- Document Ordering and Result Set

2.1.Overview

As mentioned previously, any data can be stored in the SIETS system and then retrieved using FTS queries. Data are stored in the SIETS storage as SIETS documents. A SIETS document is the smallest unit in the SIETS storage against which searching is performed. When a search request is submitted, the SIETS server searches within the SIETS storage and finds all documents that match the query.

Abstracting from specific content, format, and structure, we assume that data in existing corporate filings, databases, or storages can be perceived as documents that each have a unique ID, title, and a content consisting of textual and possibly XML marked up information in which FTS can be performed.

An ID can be a simple integer, an alphanumeric character string, a full file path on a file server and the file name, a URL of a Web page, or any other element that uniquely identifies a document.

Often there are also other elements; however, we will talk about them later. Also we assume that when performing a search request, what a user expects to have as a reply is a list of IDs, titles and short descriptions of those documents, which match the search request.

The SIETS system supports the assumed default elements for importing and retrieving data.

The following sections describe how documents are imported in the SIETS system if they are not XML structured and if they are XML structured.

2.2.Creating Document Structure with Application

When importing data to the SIETS storage using the SIETS API functions, the default document elements: ID, title, and content, are passed to the SIETS server as parameters of respective functions. When calling respective SIETS API command for storing a document in the SIETS storage, elements of the document are enclosed in XML tags and sent to the SIETS storage.

The following figure illustrates this process:

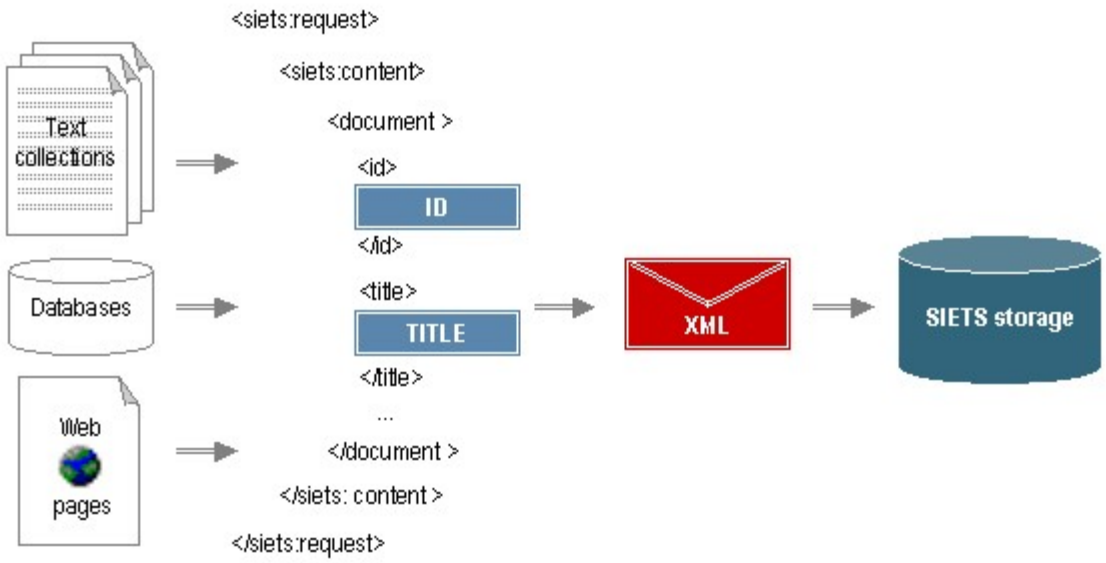

Figure 10: Storing data in the SIETS storage

The following table lists and describes all default elements for a SIETS document.

Element	Description
ID	Unique document identifier in which FTS is not performed.
Title	Document title in which FTS can be performed.
Rate	Value of the integer type in which FTS is not performed assigned to a document with a respect to other documents. When performing a search request, search results will be ordered by rate, if not by relevance. For more information on document ordering, see Document Ordering in Result Set.
Domain	Document domain. This element can be used to denote a domain of a Web document, as well as, it can be used as a classifier for any kind of documents. When performing a search, it is possible to limit the number of documents from one domain in the search result.
Text	Textual information in which FTS can be performed. SIETS also supports XML marked up information and preserves the markup, when searching in it. A snippet, which is a fragment with an occurrence of the search term, is returned to the search results.
Hidden	Textual information in which FTS can be performed, but for which a snippet is not returned to the search results.
Info	Additional information added to a document, but in which FTS is not performed, for example, picture files, MS Word or PDF document files, and so on. Note that these files must be appropriately formatted. For information on appropriate formatting, see Formatting XML Special Characters.

Extracting and defining these elements from source data before importing the data to the SIETS system is an application task.

2.3.Importing XML Structured Data

If source data are XML structured, it is not necessary to restructure it to the default SIETS document structure described in the previous section. SIETS uses the document structure

definition mechanism, called scheme, to define the location and behavior for each document part. Before you can store the XML structured source data to the SIETS storage, the scheme for the SIETS storage must be defined. The existing scheme is retrieved and a new scheme is set to the SIETS storage by calling the SIETS API commands get_scheme and set_scheme, respectively, which use XPath notation to define a location and to assign one or more policies to each document part.

For more information on the Xpath notation, see http://www.w3.org/TR/xpath.

By policy we understand a set of operations for data importing and retrieving to the SIETS storage. All policies apply to all document parts. However, each policy has a set of values, which define, to what extend does the policy apply to the document part. Each policy can have a different value set for the particular document part, for example, the policy id=no, which means that information of this document part will not be considered as the document identification part, and the policy index=all, which means that information of this document part will be indexed both: as textual information and also as textual information with preserved XML markup.

The following table lists all policies with their values. The first value listed for a policy is the default value, in other words, the value that are set if the policy is not specified for the document part.

Policy	Value	Description
id	no (default)	Information within this part will be not considered as identifier of the document. The policy is not applied to this document part.
	yes	Information within this part will be considered as identifier of the document.
rate	no (default)	Information within part will be not considered as rate of the document.
	yes	An integer number within this part will be considered as rate of the document.
domain	no (default)	Information within this part does not denote a domain of a Web document, or any other classifier of a document.
	yes	Information within this part is denotes a domain of a Web document, as well as, a classifier for any kind of documents.
index	no (default)	Information within this part will be stored in the document repository and available for retrieval, however, it will be not indexed in the inverted index.
	text	Textual information contained within this part is added to the inverted index and made available for FTS.
	xml	Textual information contained within this part preserving XML markup is added to the inverted index. In this case FTS will be performed according to the XML markup.
	all	The two above applies to this document part. It consumes more resources of memory and longer indexing time.
	classify	This index type is used for categorizing documents in some type of hierarchy, for example directory structure. Data later can be accessed using XPath expressions, relative to this part. Only one part can be set as index classify for document. See more information in chapter on XML drilldown.

Policy	Value	Description
weight	<min–max>	This policy works only together with the index policy with values: text, xml, or all. The range is from 1 to 100. All words contained in this part are explicitly set to be relevant to corresponding search term when performing FTS.
list	no (default)	Information within this part will be not listed in the search results.
	yes	Information within this part will be listed in the search results.
	highlight	Information within this part will be listed in the search results, but the search terms within this part will be highlighted.
	snippet	In the search results, from this part only a snippet will be shown. The search terms within this part will be highlighted.

Technically, there are two ways, how to set policy values for document parts:

- by calling the SIETS API command set_scheme, which sets the policy value for the document part for all documents in the SIETS storage
- by adding the siets:policy attribute to a document part tag element, for example, <title siets:index="text">, which sets the policy value for the document part for a particular document

For more information on the get_scheme and set_scheme commands, see Get_scheme and Set_scheme.

It is suggested to assign policy values by calling the SIETS API commands get_scheme and set_scheme as this option is easier and faster.

However, adding the siets:policy attribute to a document part tag element is more powerful in cases when each document in the SIETS storage you want to define a different policy values for the document parts. For example, for one document the index policy can be set to all, while for other documents in the same SIETS storage the index policy can be set to text.

These two mechanisms can be combined, for example, you can store all documents with a single policy value for the document part to the SIETS storage, and then for some documents from the SIETS storage add the siets:policy attribute to a different value.

In SIETS future releases the list of predefined policies can be expanded.

2.4.Document Ordering in Result Set

This section describes how documents are ordered in a result set. It describes the two mechanisms conceptually and contains the following topics:

- Overview
- Rate
- Relevance

2.4.1.Overview

There are two mechanisms in the SIETS system how documents are ordered in a result set:

- By rate, which must be assigned by the application to each document before storing it to the SIETS storage and is independent from a search request.

- By relevance, which is calculated when performing a search and which ensures that documents that are closer to a search request, are displayed first in a result set.

The rate ensures high performance of the search function, the relevance ensure the quality of the search results. Sorting by rate is a default mechanism that is applied every time a search is performed. Sorting by relevance is an option that you can choose additionally when a search is performed.

The decrease of performance due to the relevance is minimal.

The rate and relevance mechanisms are illustrated by an example in the following figure:

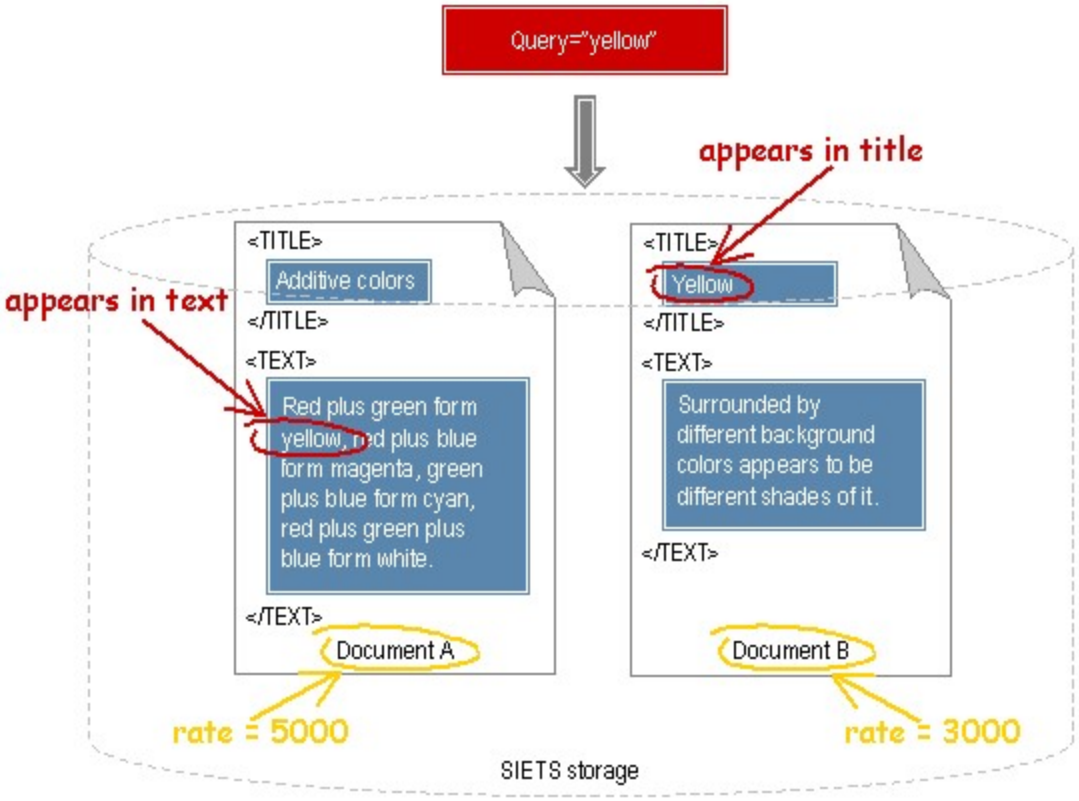

Figure 11: Document rate and relevance

The query contains the search function that must return documents containing the word yellow.

Each document in the SIETS storage has the rate assigned: the document A has a `rate=5000`, and the document B has the `rate=3000`.

The following table presents the sequence of the two documents in the result set, when the search function uses the relevance for document ordering, and when it does not, in other words, when the relevance is on and when the relevance is off.

Document and its rate	Relevance off	Relevance on
Document A rate=5000	1	2
Document B rate=3000	2	1

When searching with the relevance off, only the document rate is considered, and, therefore, documents with higher rates are displayed first. In the example, the document A has a higher rate than the document B, and, therefore, the document A is displayed first.

When searching with the relevance on, place where the search term appears in the document is considered, and, therefore, documents that contain the search term in parts that are more important than other parts, in other words, have a higher specific weight, are displayed first. In the example, the document A contains the search term in its text part, whereas the document B contains the search term in the document title, which has the higher specific weight than the text. Therefore, the document B is displayed first.

2.4.2.Rate

The rate is a number of the integer type in the range from 0 to 4294967295=2^{32}-1, which must be assigned by the application to each document when storing it to the SIETS storage.

The rate allows significant optimizations for large data amounts, which ensures high performance of the SIETS system.

It is an application developer's task to create an effective algorithm for assigning rate to document collections that is appropriate and satisfies user needs, for example, alphabetic order, by document publication or creation date, or objective document importance.

If the rate is not assigned or if there are documents with the same rate, the default document order in a result set is a reverse of the document storing sequence to the SIETS storage.

In a single SIETS storage, only one rate-assigning algorithm can be used.

If your application requires several ordering types for one document collection, then you must create several SIETS storages, which each contains the document collection with its own rate-assigning algorithm.

Technically, assigning the rate to documents is setting an integer value for the `rate` element. For more information on the SIETS document structure, see Creating Document Structure with Application.

2.4.3.Relevance

The relevance is a number of the integer type, that is a measure of the accuracy of the search results, which is calculated according to:

1. the specific weight interval of the document part in which the search term appears
2. the number of times the search term appears compared to other documents
3. the distance between the search terms in the document, if multiple words are being searched

A document part with a higher specific weight interval than other document parts mean that this part is considered as more important than the other parts. For example, the document title is more important than the document text.

In the SIETS system, there is a relevance calculation algorithm, which is implemented according to the three items described above in this section.

However, the first item: the specific weight interval can be customized to best reflect your document structure.

Fore more information on the SIETS relevance calculation algorithm, see Relevance Calculation Algorithm.

Fore more information on setting your own specific weight, see Customizing Specific Weight Interval.

2.4.3.1.Relevance Calculation Algorithm

This section describes general principles of the SIETS relevance calculation algorithm.

Note: This section contains some of SIETS system implementation details. Description provided in this section is very general and does not include implementation details for all SIETS functionality.

The SIETS relevance calculation algorithm consists of two parts that are performed when:

- storing documents to the SIETS storage
- searching documents in the SIETS storage

Steps of the SIETS relevance calculation algorithm are described generally. To ensure a better understanding of the algorithm, an example is also provided. Each step is followed by the example part that reflects the step.

1. When storing documents to the SIETS storage, specific weight for each word in a document is calculated as follows:
   1.1 In each document part, the specific weight is calculated for each word according to the specific weight interval of the document part the word occurs.

   The specific weight for a word in a document part is the minimum value of the following:
   - minimum value of the specific weight interval of the document part plus a number of times the word occurs in the document part
   - maximum value of the specific weight interval of the document part

   Note: The specific weight interval minimum and maximum can be the same value. In that case, for all words in such document part, no matter how often they appear, the specific weight in the document part is the same: the specific weight value of the document part.

Example:

A document consists of three document parts: heading, description, and note. Each document part contains words w_1, w_2, and w_3 and has its own specific weight interval, as described in the following figure:

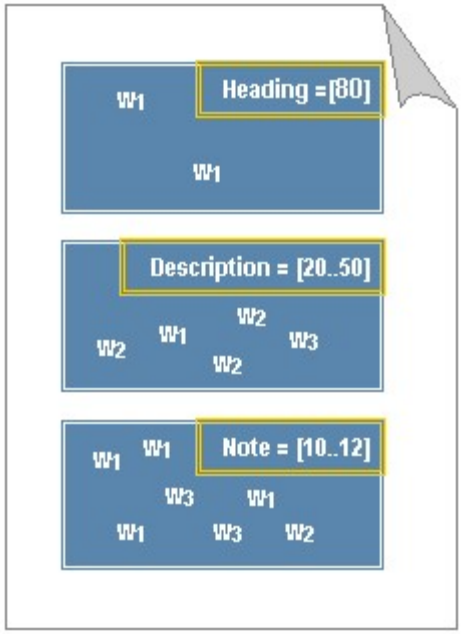

Figure 12: Calculating specific weight for each document

w_1(heading)=min(80,80)=80, w_1(description)=min(20+1,50)=21, w_1(note)=min(10+4,12)=12

w_2(heading)=0, w_2(description)=min(20+3,50)=23, w_2 (note) min(10+1,12)=11

w_3(heading)=0, w_3(description)=min(20+1,50)=21, w_3 (note) min(10+2,12)=12

1.2 The maximum value of specific weights of a word in all document parts is assigned as the specific weight of the word in the document.

Example (continued):

max(w_1(heading), w_1(description), w_1(note))=80

max(w_2(heading), w_2(description), w_2(note))=23

max(w_3(heading), w_3(description), w_3(note))=21

2. When searching documents in the SIETS storage, the relevance of the document according to the search request is calculated as follows:

2.1 Specific weights of all search terms in a document are summed.

Example (continued):

$\Sigma(w_1, w_2, w_3)$ = max(w_1(heading), w_1(description), w_1(note)) + max(w_2(heading), w_2(description), w_2(note)) + max(w_3(heading), w_3(description), w_3(note)) = 124

2.2 The relevance is calculated by multiplying the sum from the previous step with a value that is calculated taking into the account the distance between the search terms in the document: the greater the distance, the smaller the value

Example (continued):

Relevance = $\Sigma(w_1, w_2, w_3)$ * d

2.4.3.2.Customizing Specific Weight Interval

This section describes how to set specific weight interval for document parts that best reflects your document structure.

As described in the previous section, a specific weight interval for a document part is an interval between two integer numbers.

By default, the following specific weights are defined:

Document part	Minimum	Maximum
Title	100	100
All except Title	1	99

You can set a different value for the title part, and you can define a separate specific weight interval for each document part, such as Text and Hidden, or other document parts that you have, to ensured more detailed relevance calculation.

Because of the performance considerations, there is a limit for the maximum specific weight interval value, which is 255.

Technically, there are two ways, how to customize specific weight intervals for document parts:

- by calling the SIETS API command set_scheme, which sets the specific weight interval value for the document part for all documents in the SIETS storage
- by adding the siets:weight attribute to a document part tag element, for example, <title siets:weigth="75">Title text</title>, which sets the specific weight interval value for the document part for a particular document

For more information on the get_scheme and set_scheme commands, see Get_scheme and Set_scheme.

It is suggested to assign specific weight interval by calling the SIETS API commands get_scheme and set_scheme as this option is easier and faster.

However, adding the siets:weight attribute to a document part tag element is more powerful in cases, when for each document in the SIETS storage you want to define a different set of specific weight intervals for the document parts. For example, for one document the specific weight interval of the title part can be set to 100, while for other documents in the same SIETS storage the specific weight interval of the title part can be set to 80.

These two mechanisms can be combined, for example, you can store all documents with a single set of specific weight interval to the SIETS storage, and then for some documents from the SIETS storage add the siets:weight attribute to a different value.

3. INTERNATIONALIZATION

This section describes multi-language support and character encoding concepts, provides examples for different character encoding cases, and explains XML formatting concepts.

This section contains the following topics:

- Multi-language Support and Text Encoding
- Formatting XML Special Characters

3.1.Multi-language Support and Character Encoding

The SIETS API structure is based on XML, which means that all character encoding related issues adhere XML internationalization standards.

For more information on XML internationalization standards, see http://www.w3.org/TR/REC-xml.

SIETS provides a complete multi-language support, by automatically performing all necessary character encoding conversions.

You can import documents in different languages and encodings in a single SIETS storage, as well as you can perform search queries in different languages and encodings in a single SIETS storage.

This section contains the following topics:

- Overview
- Storing and Searching in a Single Encoding
- Storing in Different Encodings and Searching in a Multiple Bytes per Character Encoding
- Storing in Different Encodings and Searching in One Byte per Character Encoding

3.1.1.Overview

For document importing and searching SIETS supports any language and text encoding. When importing documents to the SIETS storage, internally, all documents are converted and stored in the UTF-8 encoding, as illustrated in the following figure:

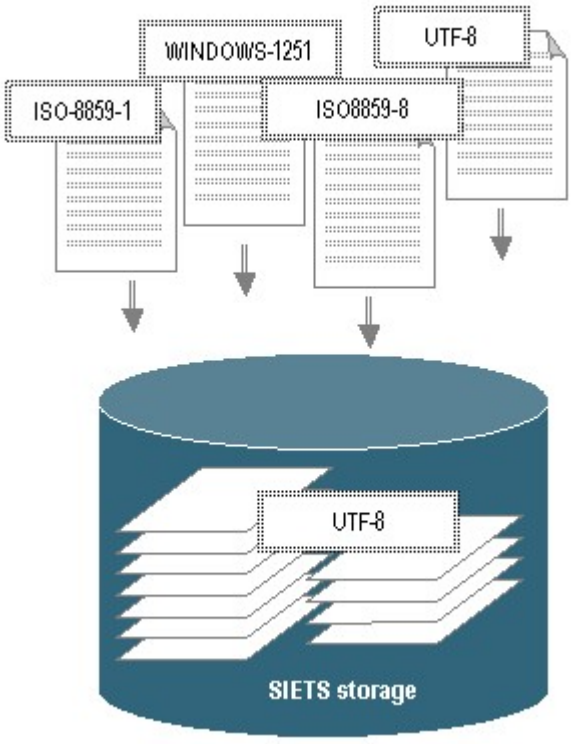

Figure 13: Importing documents with different encodings

In Figure 13, document encodings are represented as encoding values each in a white box with a double dotted border.

Data exchange between an application and the SIETS server is performed in the XML format. In the XML format, data can be in any encoding; this encoding is defined in the XML header of the document.

All SIETS API functions have an encoding parameter, which defines the encoding of textual data. This encoding is used in the XML header, when importing documents to the SIETS storage as described in Creating Document Structure with Application. The textual data must comply with the encoding defined in a function parameter, or else the system returns a parsing error.

The number of encodings is only limited to those that are installed on a computer on which the SIETS server is run. To find out what encodings are installed on the SIETS server computer, see the *SIETS Administration and Configuration Guide*. For example, on RedHat Linux, usually, US-ASCII, ISO8859-1..13, WINDOWS-1250..1258, UTF-7, UTF-8, UTF-16, and UTF-32 encodings are installed.

Technically, only the encoding is important to the SIETS system, which means that you can store and search data in the SIETS system in any language as long as you supply a valid encoding for that language.

There are the following two types of encodings:

Title	Description
one byte per character	Contains 256 characters, which means that, within one such encoding, characters for several similar languages can be included, for example, WINDOWS-1250 and ISO8853.
multiple bytes per character	Contains all UCS (universal character set) characters, which include characters for almost all languages, for example, Greek, Cyrillic, Korean, and so on.

You can store documents in different languages with different encodings within a single SIETS storage; documents are converted to the UTF-8 encoding, which contains all characters from UCS and, therefore, all characters are preserved correctly.

Search results are returned in the encoding that is used for the search request.

The following three sections contain examples with different cases of working with a single and several encodings, which demonstrate the SIETS multi-language support.

3.1.2.Storing and Searching in Single Encoding

This section contains an example, when a single encoding is used for document storing and retrieving.

The following figure illustrates the example:

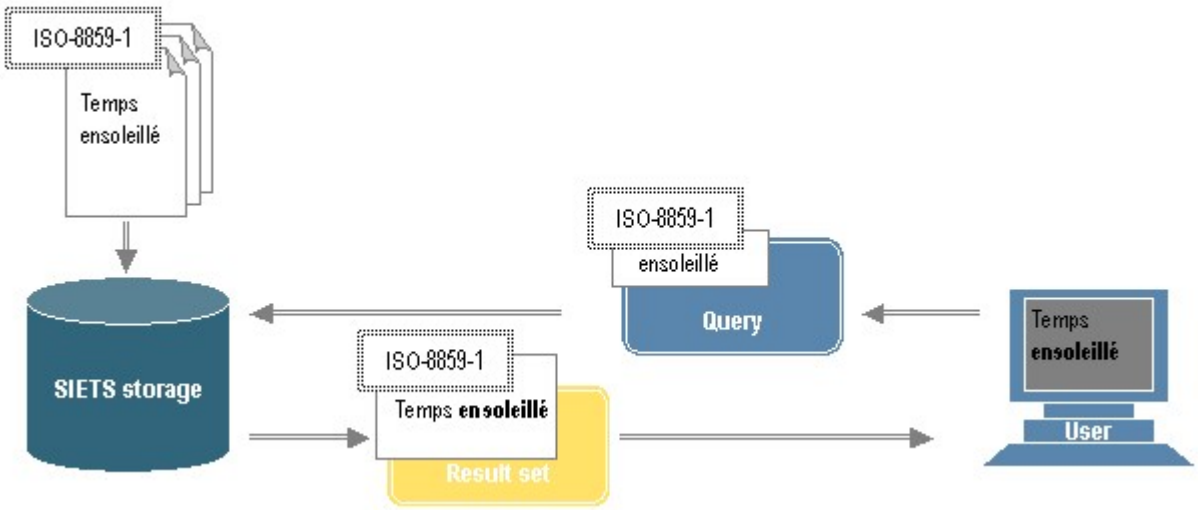

Figure 14: Storing and searching in single encoding

In Figure 14, document encodings are represented as encoding values each in a white box with a double dotted border.

1. All documents are imported to the SIETS storage in the same encoding. In Figure 14, the encoding is ISO-8859-1 for French, which encodes French character é and other characters that are not included in a US-ASCII encoding.
2. Users submit search queries to the SIETS storage in the same encoding as the document source encoding.
3. Search results are returned to a result set in the same encoding.

4. The search results are displayed with correct characters to the user.

Note: The user computer must have appropriate fonts installed for viewing that encoding. Older browser versions may not support the UTF-8 encoding and display the special characters as question marks ?. In that case, the browser must be updated.

3.1.3.Storing in Different Encodings and Searching in Multiple Bytes per Character Encoding

This section contains an example, when different encodings are used for document storing and a multiple bytes per character encoding is used for retrieval.

The following figure illustrates the example:

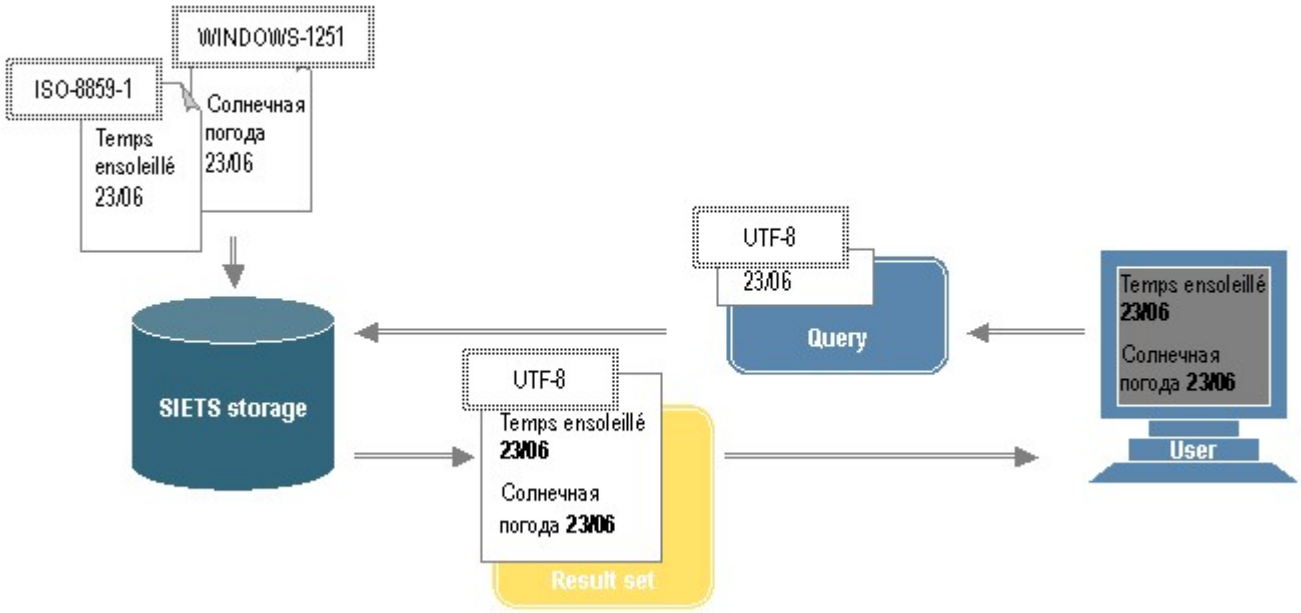

Figure 15: Storing in different encodings and searching in multiple bytes per character encoding

In Figure 15, document encodings are represented as encoding values each in a white box with a double dotted border.

1. Documents are imported to the SIETS storage in different encodings. In Figure 15, the encodings are ISO-8859-1 for French and WINDOWS-1251 for Russian.
2. Users submit search queries to the SIETS storage in a multiple bytes per character encoding, in Figure 15, the encoding is UTF-8.
3. Search results are returned to a result set in the encoding, which is used in the search request, in Figure 15, the encoding is UTF-8.

As in this case the multiple bytes per character encoding is used, there are no problems for displaying characters for both languages.

4. The search results are displayed with correct characters to the user.

Note: The user computer must have appropriate fonts installed for viewing that encoding. Older browser versions may not support the UTF-8 encoding and display the special characters as question marks ?. In that case, the browser must be updated.

3.1.4.Storing in Different Encodings and Searching in One Byte per Character Encoding

This section contains an example, when different encodings are used for document storing and a one byte per character encoding is used for retrieval.

The following figure illustrates the example:

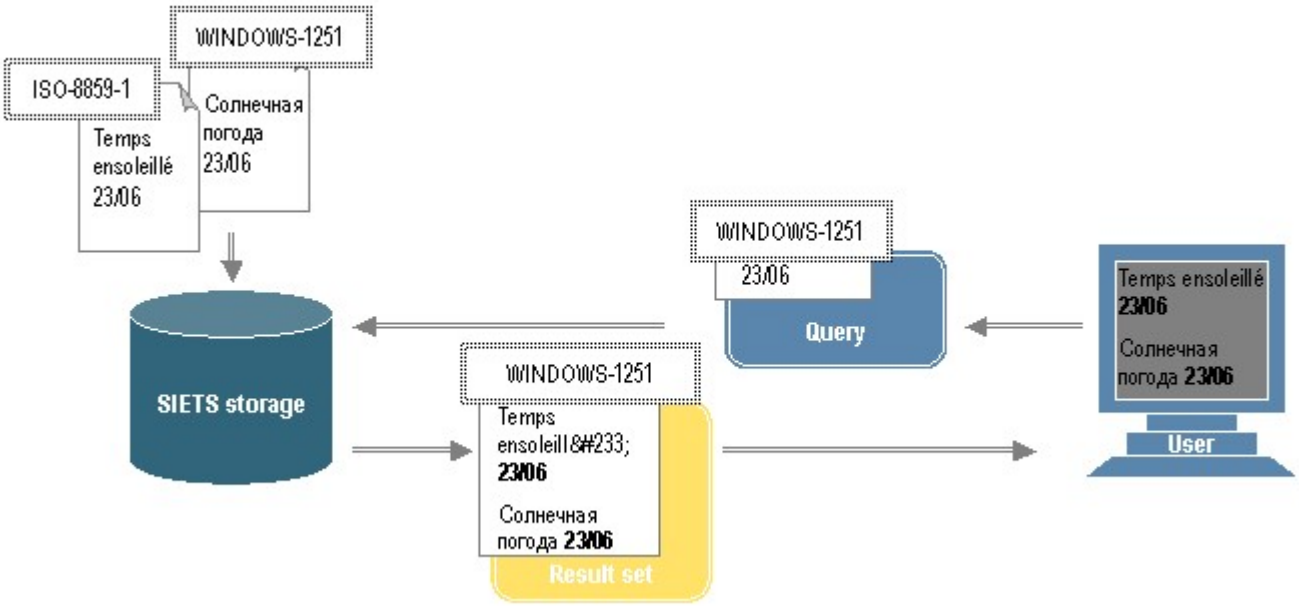

Figure 16: Storing in different encodings and searching in one byte per character encoding

In Figure 16, document encodings are represented as encoding values each in a white box with a double dotted border.

1. Documents are imported to the SIETS storage in different encodings. In Figure 16, the encodings are ISO-8859-1 for French and WINDOWS-1251 for Russian.
2. Users submit search queries to the SIETS storage in a one byte per character encoding, in Figure 16, the encoding is for Russian.
3. Search results are returned to a result set in the encoding, which is used in the search request, in Figure 16, the encoding is for Russian.

Characters that are not in the encoding are returned as XML entities, in Figure 16, the French symbol é is returned as `&#233;`.

For more information on XML entities, see http://www.w3.org/TR/REC-xml.

4. The search results are displayed with correct characters to the user.

Note: The user computer must have appropriate fonts installed for viewing that encoding. Older browser versions may not support the UTF-8 encoding and display the special characters as question marks ?. In that case, the browser must be updated.

3.2.Formatting XML Special Characters

As mentioned in earlier sections, data are sent from an application to the SIETS storage in the XML formatting. Therefore, the data must comply with XML formatting rules, for example, the data cannot contain XML special characters like <, >, and &, which are used for the XML markup, instead, <, >, and &amp must be used respectively.

Example:

If you have a title A&B, you must convert it to A&B.

For more information on the XML formatting rules, see http://www.w3.org/TR/REC-xml.

4. SIETS API SPECIFICATION

This section generally describes all SIETS API specification, which is implemented in XML.

This section contains the following topics:

- Overview
- SIETS XML Message Envelope
- Data Manipulation
- Status Monitoring
- Data Retrieval
- Error Handling

4.1.Overview

This section contains the following topics:

- Exchanging Messages
- XML Message Structure

4.1.1.Submitting SIETS Commands and Receiving Replies

XML request and reply messages are exchanged between the application and the SIETS storage via HTTP with the port 80 as the default.

As mentioned earlier, it is possible to transport SIETS commands to the SIETS server and receive replies as XML messages and, also it is possible to submit HTTP GET parameters and receive formatted replies.

Both options are described in the following sections:

- Exchanging XML Messages Directly
- Submitting Parameters and Receiving Formatted Replies

4.1.1.1.Exchanging XML Messages Directly

The following figure illustrates submitting SIETS commands and receiving replies via XML messages directly:

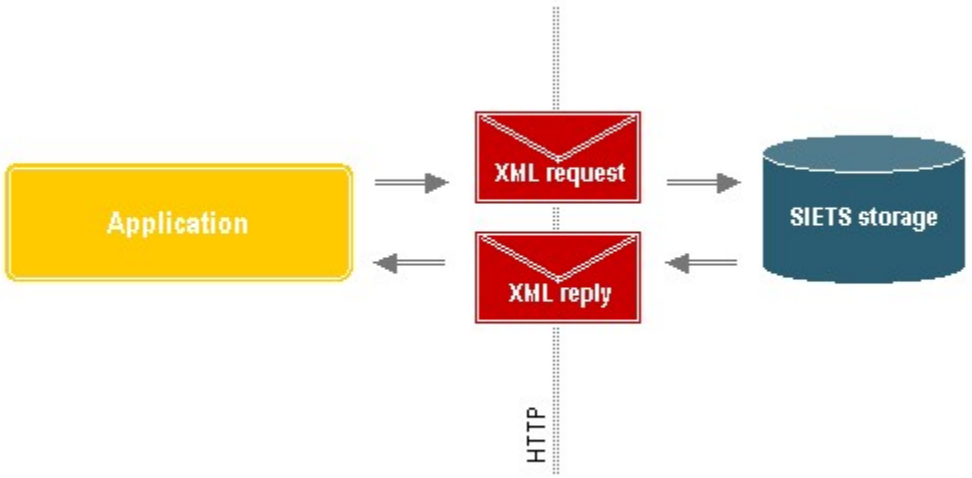

Figure 17: Exchanging XML messages directly

A request is sent as a POST method.

As the HTTP resource identification, the URL http://host/cgi-bin/siets/api.cgi must be used, where <host> is the SIETS server host name.

4.1.1.2.Submitting Parameters and Receiving Formatted Replies

The following figure illustrates submitting SIETS commands as HTTP GET parameters and receiving formatted XML replies:

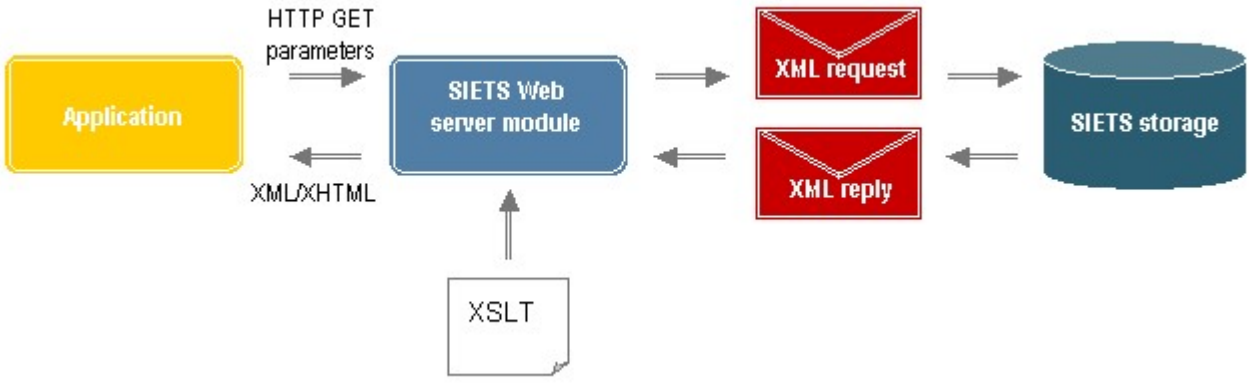

Figure 18: Submitting parameters and receiving formatted replies

A request is sent as a GET or POST method.

As the HTTP resource identification, the URL http://host/cgi-bin/siets/api.cgi must be used, where <host> is the SIETS server host name. Command specific parameters must be included in query string or passed as POST data.

4.1.2.XML Message Structure

As described previously, each XML message contains a command name, content data that are specific for the command, and other information, such as user name and request identifier, which is common for all XML messages and included in the so called XML message envelope.

For more information on the XML message envelope, see SIETS XML Message Envelope.

The following figure illustrates the common part for all XML messages and content part that is specific for each command:

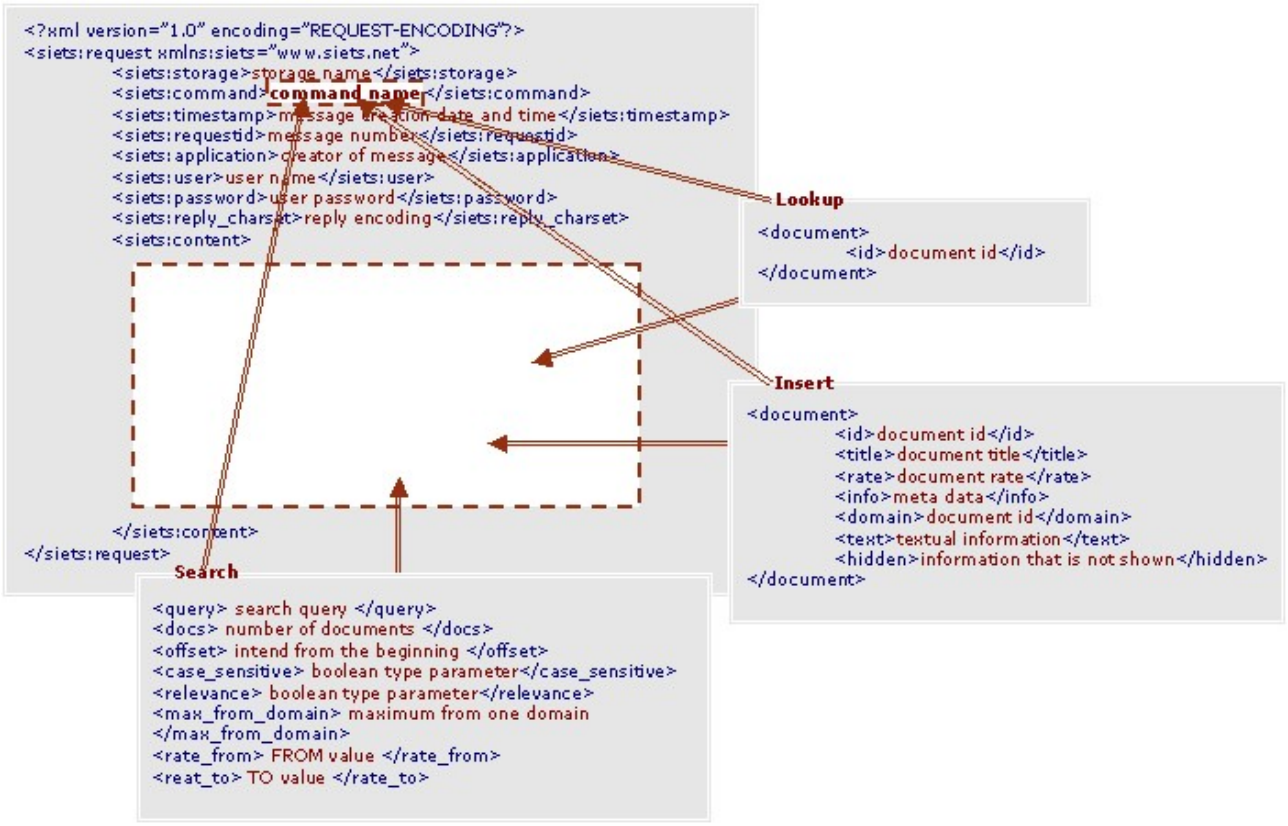

Figure 19: XML message: common part and content part

Description of SIETS API commands is organized so that the common part is described in SIETS XML Message Envelope, and only the content parts are described for each command in separate sections named after the command.

XML elements are presented as they appear in messages and each XML element is described within its tags.

The command syntax consists of an XML request and an XML reply, and as mentioned, XML requests can be submitted as HTTP GET or POST parameters. To describe XML request, XML reply, and HTTP GET parameters syntax, each section contains the following subsections:

Subsection	Description
XML Request	Lists all XML request elements that specific for the command as they appear in XML request messages. Each element is described within its tags. The description within the tags ends with an asterisk *, if the element is mandatory.
HTTP GET Parameters	Describes HTTP GET parameter syntax in the form of an example. The example looks as follows: http://host/cgi-bin/siets/api.cgi?param1=value&param2=value where: • host is SIETS server IP address or a host name, • param1, param2, and so on are HTTP GET parameters, • value is a parameter's value. **Note:** In examples HTTP GET parameters are described, however, you can submit also POST parameters.
XML Reply	Lists all XML reply elements that are specific for the command as they appear in XML reply messages. Each element is described within its tags.

Some elements in XML requests, and thus, respective parameters, if submitting the XML request as HTTP GET parameters, are mandatory, and some are not. The mandatory elements are marked with an asterisk * in the XML request description.

However, there are some XML request elements that are mandatory only if submitted as XML request, but are not mandatory if submitted as HTTP GET parameters. Such parameters first must be defined in the SIETS Web server module configuration file, and then, do not have to be submitted each time when sending a command. Parameters that can be defined in the SIETS Web server module configuration file are the following:

- user name
- user password

For more information on the SIETS Web server module configuration file, see the *SIETS Administration and Configuration Guide.*

4.2.SIETS XML Message Envelope

This section describes the common parts of the XML request and reply for all SIETS API commands.

4.2.1.1.XML Request

```
<?xml version="1.0" encoding="REQUEST-ENCODING"?>
<siets:request xmlns:siets="www.siets.net">
    <siets:storage>storage name*</siets:storage>
    <siets:command>command name*</siets:command>
    <siets:timestamp>message creation date and time</siets:timestamp>
    <siets:requestid>message number</siets:requestid>
    <siets:application>creator of message</siets:application>
    <siets:user>user name*</siets:user>
        <siets:password>user password*</siets:password>
    <siets:timeout> function timeout period </siets:timeout>
    <siets:reply_encoding>reply encoding</siets:reply_encoding>
```

```
    <siets:content>command specific data      </siets:content>
</siets_request>
```

4.2.1.2.XML Reply

```
<?xml version="1.0" encoding="REPLY-ENCODING"?>
<siets:reply xmlns:siets="www.siets.net">
    <siets:storage>storage name</siets:storage>
    <siets:timestamp>reply creation date and time</siets:timestamp>
    <siets:content>command specific data</siets:content>
    <siets:command>command name for which the reply is created</siets:command>
    <siets:requestid>message number for which the reply is created</siets:requestid>
    <siets:seconds>time period for the reply creation</siets:seconds>
    <siets:replyid>unique message id created by the SIETS server</siets:replyid>
</siets_reply>
```

4.3.Data Manipulation

This section describes the following data manipulation commands:

- Insert, Update, and Replace
- Delete
- Index
- Clear
- Get_scheme
- Set_scheme

4.3.1.Insert, Update, and Replace

The insert command adds a document to the SIETS storage. If a document with such ID already exists, the command returns an error.

If a document with such ID exists in the SIETS storage, the update command updates the document. If a document with such ID is not in the SIETS storage, the update command adds it to the SIETS storage.

The replace command replaces contents of a document in the SIETS storage. If a document with such ID is not in the SIETS storage, the command returns an error.

4.3.1.1.XML Request

```
<siets:content>
    <document>document content      <document>
</siets:content>
```

Where the document content consists of document structure elements. The default SIETS document structure is as follows:

```
<document>
    <id> document id * </id>
    <title> document title </title>
    <rate> document rate </rate>
    <domain> document domain  </domain>
    <info> meta data </info>
    <text> textual information, which is used for indexing </text>
```

```
    <hidden> textual information, which is used for indexing, but which is not
shown</hidden>
</document>
```

For more information on the default SIETS document structure, see Creating Document Structure with Application.

4.3.1.2.HTTP GET Parameters

```
http://host/cgi-bin/siets/api.cgi?command=insert&storage=test&id=1&title=Doc1
http://host/cgi-bin/siets/api.cgi?command=update&storage=test&id=1&title=Doc1
http://host/cgi-bin/siets/api.cgi?command=replace&storage=test&id=1&title=Doc1
```

4.3.1.3.XML Reply

If the command is executed successfully, the XML reply does not contain any command specific data.

If the command is not executed successfully, an error is returned. For more information on errors, see Error Handling.

4.3.1.4.Binary Files Conversion

Note: The binary files conversion functionality is available only starting from the SIETS server version 3.2.8.

Binary files conversion is integrated feature for the insert, update, and replace commands. The binary files conversion functionality converts binary file contents into plain text. Thus, it is possible to add several Microsoft Office files and other binary files to the SIETS storage and perform full text search on them.

The following table lists extensions of binary files that can be added to the SIETS storage:

Extension	Description
DOC	Microsoft Word document.
XLS	Microsoft Excel document.
PPT	Microsoft PowerPoint document.
RTF	Rich text format document.
PDF	Adobe portable document format document.
PS	Post script document.

To use the binary files conversion functionality, in the XML request, in the place of the text tag, use the file tag in the following format:

```
    <file store=”yes/no” <!--If store=”yes”, then the original document is stored in the
SIETS storage and returned when retrieved. The default value is “no”-->>
        <ext> extension of binary file </ext>
        <data> data of binary file converted to the base64 encoding </data>
    </file>
```

As described in the data tag, binary file contents first must be converted to the base64 encoding. This is because XML does not support storing binary data within a tag.

4.3.2.Delete

The delete command deletes a document from the SIETS storage. If a document with such ID is not in the SIETS storage, the command returns an error.

4.3.2.1.XML Request

```
<siets:content>
    <document>
        <id>document id *</id>
    </document>
</siets:content>
```

4.3.2.2.HTTP GET Parameters

```
http://host/cgi-bin/siets/api.cgi?command=delete&storage=test&id=1
```

4.3.2.3.XML Reply

If the command is executed successfully, the XML reply does not contain any command specific data.

If the command is not executed successfully, an error is returned. For more information on errors, see Error Handling.

4.3.3.Index

After inserting, updating, replacing, or deleting documents in the SIETS storage, the SIETS server must permanently save the changes made to the inverted index. The SIETS server is able to make the decision, when to start saving the changes to the inverted index, on its own. However, to optimize performance, for large data amounts, it is recommended to inform the system when a portion of documents are loaded and in the nearest time period more documents are not to be loaded, in other words, the SIETS server can allocate all resource for the process of indexing.

The index command tells the SIETS server to start the process of indexing.

4.3.3.1.XML Request

The <siets:content> element does not contain any command specific data.

4.3.3.2.HTTP GET Parameters

```
http://host/cgi-bin/siets/api.cgi?command=index
```

4.3.3.3.XML Reply

If the command is executed successfully, the XML reply does not contain any command specific data.

If the command is not executed successfully, an error is returned. For more information on errors, see Error Handling.

4.3.4.Clear

The clear command deletes all documents from the SIETS storage. This command should be used only when a complete re-indexing of the SIETS storage is necessary.

4.3.4.1.XML Request

The <siets:content> element does not contain any command specific data.

4.3.4.2.HTTP GET Parameters

```
http://host/cgi-bin/siets/api.cgi?command=clear
```

4.3.4.3.XML Reply

If the command is executed successfully, the XML reply does not contain any command specific data.

If the command is not executed successfully, an error is returned. For more information on errors, see Error Handling.

4.3.5.Get_scheme

The get_scheme command retrieves the document structure definition, in other words, scheme, from the SIETS storage.

Note: It is also possible to review and edit the document policy scheme from SIETS Enterprise Manager. For information on SIETS Enterprise Manager, see the *SIETS Administration and Configuration Guide,* Configuring SIETS Storage.

4.3.5.1.XML Request

The <siets:content> element does not contain any command specific data.

4.3.5.2.HTTP GET Parameters

```
http://host/cgi-bin/siets/api.cgi?command=get_scheme
```

4.3.5.3.XML Reply

```
<siets:content>
    <scheme>
        <part>
                <location>location of the document part in XPath notation</location>
                <policy>policy for the document part, policy=value </policy><!--this
element is repeated for each policy that the document part has-->
        </part><!--this element is repeated for document part that the document has-->
    </scheme>
</siets:content>
```

4.3.6.Set_scheme

The set_scheme command sets the document structure definition, in other words, scheme, to the SIETS storage.

Note: If you modify the scheme, it applies to all documents that are to be imported to the SIETS storage. However, it does not automatically modify the document structure for documents that already are imported to the SIETS storage.

Note: It is also possible to review and edit the document policy scheme from SIETS Enterprise Manager. For information on SIETS Enterprise Manager, see the *SIETS Administration and Configuration Guide.*

4.3.6.1.XML Request

```
<siets:content>
    <scheme>
        <part>
                <location>location of the document part in XPath notation</location>
                <policy>policy for the document part, policy=value</policy><!--this element
is repeated for each policy that the document part has-->
        </part><!--this element is repeated for document part that the document has-->
    </scheme>
</siets:content>
```

For more information on document policies, see Importing XML Structured Data.

4.3.6.2.HTTP GET Parameters

The set_scheme command cannot be submitted as HTTP GET parameters.

4.3.6.3.XML Reply

If the command is executed successfully, the XML reply does not contain any command specific data.

If the command is not executed successfully, an error is returned. For more information on errors, see Error Handling.

4.4.Status Monitoring

This section describes the Status command.

4.4.1.Status

The status command returns status information of the SIETS server instance. The status information includes:

- number of documents in the SIETS storage
- number of words in the vocabulary
- total number of words in the SIETS storage
- number of executed commands since the last startup of the instance
- number of errors that have occurred since the last startup of the instance

4.4.1.1.XML Request

The <siets:content> element does not contain any command specific data.

4.4.1.2.HTTP GET Parameters

```
http://host/cgi-bin/siets/api.cgi?command=status
```

4.4.1.3.XML Reply

If the command is executed successfully, the XML reply contains the following command specific data.

```
<siets:content>
```

```
    <status>
        <ctrld>
                <started> date and time, when the SIETS server was started </started>
                <age>time period the SIETS server is working since it was started</age>
                <total_time_elapsed>total time spent by the SIETS sever executing
commands</total_time_elapsed>
                <transactions><--This element contains information about executed
commands-->
                        <total> total number of commands executed</total>
                        <successful>number of commands that were successfully
executed</successful>
                        <failed> number of commands that were executed unsuccessfully
</failed>
                        <requests command="command name">number of times the
command was executed </requests> <-- This element is repeated for every command that
was executed.-->
                </transactions>
                <last_modified> date and time, when modifications in SIETS storage
occurred last time </last_modified>
                <queue> number of commands executed simultaneously </queue>
                <version> SIETS version number</version>
        </ctrld>
        <mtxd> <-- This element contains information about the inverted index.-->
                <journal>
                        <usage> indexing memory cache usage in percent</usage>
                </journal>
                <pool_state> index state: normal, expanding, or collapsing</pool_state>
        </mtxd>
        <wordd> <-- This element contains information about the vocabulary.-->
                <unique_words>unique words in the SIETS storage</unique_words>
                <total_words>total number of all words</total_words>
        </wordd>
        <docd>
                <documents>total number of documents</documents>
                <domains> number of distinct domains of documents</domains>
        </docd>
    </status>
</siets:content>
```

When importing data to the SIETS storage:

- If the memory reserved for memory cache is enough for the data amount being imported, the index state is normal.
- If the memory reserved for memory cache is not enough for the data amount being imported, the index state is one of the following:

Title	Description
expanding	The data being imported are written to another cache, which is written to the disk.
collapsing	When the importing is complete, the SIETS server is committing data written on the disk to the SIETS storage.

Note: While the index state is expanding or collapsing, the data written to the disk are not available for FTS. Only when data are added to the inverted index, they are available for FTS.

If the command is not executed successfully, an error is returned. For more information on errors, see Error Handling.

4.5.Data Retrieval

This section describes the following data retrieval commands:

- Lookup and Retrieve
- Search
- Select
- Similar
- Alternatives
- List last

4.5.1.Lookup and Retrieve

The lookup command searches for a document in the SIETS storage and returns the information whether the document with such ID exists is in the SIETS storage or it does not.

The retrieve command returns a document from the SIETS storage. If a document with such ID is not in the SIETS storage, the command returns an error.

4.5.1.1.XML Request

```
<siets:content>
    <document>
        <id>document id *</id>
    </document>
</siets:content>
```

4.5.1.2.HTTP GET Parameters

```
http://host/cgi-bin/siets/api.cgi?command=lookup&storage=test&id=1
http://host/cgi-bin/siets/api.cgi?command=retrieve&storage=test&id=1
```

4.5.1.3.XML Reply

If the command is executed successfully, the XML reply contains the following command specific data.

```
<siets:content>
    <found>indicator 1 or 0 if a document is found or not, respectively</found>
    <results>
        <document>
                meta data for the lookup command
                textual information for the retrieve command
        </document>
    </results>
</siets:content>
```

Meta data for the lookup command is information included in tags, for which the policy list is set to YES. By default, these are id, title, and rate tags.

For more information on policies, see Importing XML Structured Data.

If the command is not executed successfully, an error is returned. For more information on errors, see Error Handling.

4.5.2.Search

The search command performs FTS in the SIETS storage.

4.5.2.1.XML Request

```
<siets:content>
    <query> search query *</query>
    <docs> number of documents in the result set </docs>
    <offset> intend from the beginning of the result set</offset>
    <case_sensitive> Boolean type parameter: YES to enable case sensitivity of the first letter of words when performing the search, NO not to enable case sensitivity </case_sensitive>
    <relevance> Boolean type parameter: YES to order results by relevance, NO not to order results by relevance </relevance>
    <max_from_domain> Maximum number of documents from one domain. Results from one domain are grouped together within one result page. If the parameter is not set, the default value is 0, which implies that no grouping by domains is performed and no limit is set. </max_from_domain>
    <rate_from> searching documents with in a rate range: the FROM value </rate_from>
    <rate_to> searching documents with in a rate range: the TO value </rate_to>
    <wildcards> <!-- This element contains parameters for configuring wildcard patterns support. Functionality of this tag is available only starting from the SIETS server version 3.2.8.-->
        <allow> Information whether the wildcard patterns search is enabled. Values “yes” or “no”.</allow>
        <cover_factor> When wildcard patterns are used to define a class of words to be searched, only a limited number of statistically frequent words are searched for to ensure a higher performance. This element defines the limit in percent from the sum of all words created from the wildcard pattern appearance in the SIETS storage.</cover_factor>
        <min_expand> The minimum limit of the wildcard patterns matching set from the SIETS storage vocabulary in absolute numbers. This parameter overcomes the cover_factor parameter. For example, if only 2 words fall in the cover_factor, but the min_exapand is 4, then 4 words are being used in the search.</min_expand>
        <max_expand> The maximum limit of the wildcard patterns matching set from the SIETS storage vocabulary in absolute numbers. This parameter overcomes the cover_factor parameter. For example, if 20 words fall in the cover_factor, but the max_exapand is 16, then only 16 words are being used in the search.</max_expand>
    </wildcards>
</siets:content>
```

If values for the wildcards tag are not defined, corresponding parameters set in the SIETS storage configuration file are used.

For more information configuring SIETS storage, see the *SIETS Administration and Configuration Guide*.

This section contains the following topics:

- Search Query Syntax
- Case Sensitivity for Proper Names
- Grouping Results by Domain
- Filtering Results by Rate
- Web Friendly Result Navigation

Search Query Syntax

SIETS provides several mechanisms for specifying your search query. Each mechanism has a definite syntax, which is described in the following subsections. For a better understanding, each subsection also contains an example of the mechanism described and an explanation about what the example search query returns.

This section contains the following topics:

- Single Search Term
- AND
- Phrase Search
- OR
- NOT
- Boolean Expressions
- Wildcard Patterns
- Ignored Words
- Stemming
- Search within Markup
- Proximity Search
- Numeric Search

Single Search Term

To search for documents that contain a single search term, the search term must be entered as is.

Example:

John returns documents that contain the word “John”.

AND

To search for documents that contain all of the several terms, but which are not necessarily next to each other, the search term must be separated by the space character.

Example:

John Smith returns documents that contain the word “John” and the word “Smith”.

Phrase Search

To search for documents that contain an exact phrase, the search phrase must be enclosed in the quotations marks.

Example:

“John Smith” returns documents that contain the exact phrase “John Smith”.

OR

To search for documents that contain any of the search terms, the search terms must be enclosed in { } and separated with the space character.

Example:

{John Smith} returns documents that contain either the word “John” or the word “Smith”.

NOT

To search for documents that do not contain the search term, the search term must be preceded with ~.

Example:

~John returns documents that do not contain the word “John”.

Boolean Expressions

AND, OR, and NOT logical connectives can be combined in more complex search expressions using the brackets ( ), which allows you to build any Boolean expression.

Example:

{(John Smith) (Abby Brown)} returns documents that either contains the word “John” and the word “Smith”, or the word “Abby” and the word “Brown”.

{(A B ~C) ″D E″} is parsed in the expression tree as follows:

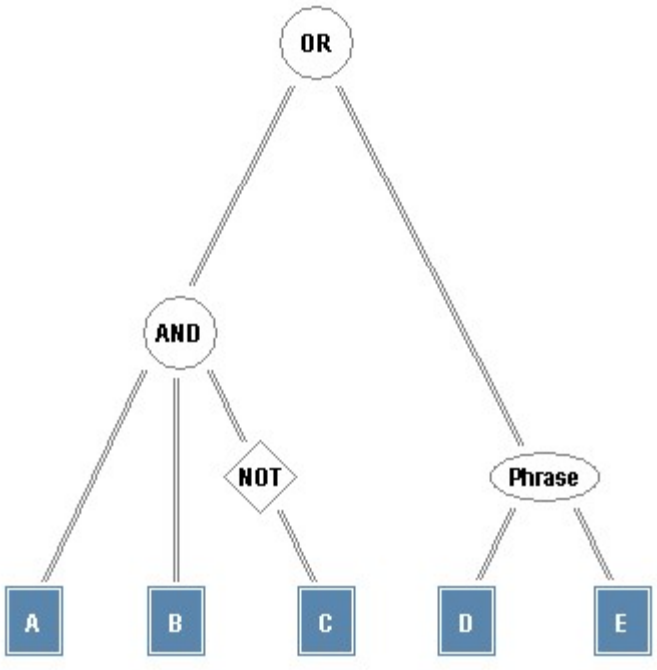

Figure 20: Search query expression tree

Wildcard Patterns

To search for documents that contain a class of words represent:

- exactly one unknown character using the question mark ?
- one or more unknown characters using the asterisk *
- range of definite characters for one unknown character occurrence using the square brackets [ ]

Note: When wildcard patterns are used to define a class of words to be searched, only a limited number of statistically frequent words are searched for. This limitation is introduced to preserve the high performance of the SIETS server. However, the maximum number of the words being searched can be increased or decreased, when configuring the SIETS server. For more information on configuring the SIETS server, see the *SIETS Administration and Configuration Guide*.

Example:

`ca?` returns documents that contain the word “car”, “cat”, “cap”, “can”, and so on.

`Joh*` returns documents that contain the word “John”, “Johnson”, “Johnny”, and so on.

`ca[pt]` returns documents that contain only the word “cap” or “cat”.

`c?[au]*` returns documents that contain the word “counter”, “club”, “chapter”, “country”, “change”, “chat”, “council”, “class”, “cpu”, “challenge", “church”, “couple”, “championship”, and so on.

Ignored Words

By default, SIETS ignores common words and characters such as “and”, ”where”, and “how”, as well as certain single characters and single letters, because they tend to slow down the search without improving the search results. Common words and characters like this are called ignored words.

The SIETS server detects words that appear in the SIETS storage most often and adds them to the ignored words list. It is possible to edit the limit of the ignored words list. For more information on managing the ignored word list limit, see the *SIETS Administration and Configuration Guide*.

If a common word or a character is essential to getting the results you want, you can include it by preceding it with a plus sign +.

Example:

`John +and Abby` returns documents that contain all three words: “John”, “and”, and “Abby”.

Stemming

It is possible to include in one search request a word and its declinations, for example, “go” and “going”.

This feature is especially useful for so-called synthetic languages, in which syntactic relations within sentences are expressed by the change in the form of a word that indicates distinctions of tense, person, gender, number, mood, voice, and case, for example, German and Latin.

To enable the declination search, a shared library must be implemented, which exports a function that extracts word roots.

For information on installing the shared library for the SIETS server, see the *SIETS Administration and Configuration Guide*.

To search for documents that contain words in declinations, a word or a phrase must be enclosed in the dollar signs $ $.

Example:

$John$ returns documents that contain the word “John” and “John’s”.

Search within Markup

To search for documents that contain the search term in a specific tag, the search term must be enclosed in the appropriate tags.

Note: The searching within markup can be performed only if the policy index with values xml or all is used. For the default document structure is the index policy with the value xml is set by default. For more information on policies see, Importing XML Structured Data.

Example:

<person>John Smith</person> returns documents that contain the word “John” in the <person> tag and the word “Smith” in the <person> tag.

{<person>John</person> <address>”New York”</address>} returns documents that either contains the word “John” in the <person> tag, or the phrase “New York” in the <address> tag”.

Proximity Search

It is possible to define maximum of words, which appear between certain search terms. These search terms are also defined in the search query. Such feature is called proximity search.

To use the proximity search feature, the search term must be as follows:

@ N term1 term2 @,

where N is the maximum count of words between the search terms, and term1 and term2 are search terms. There can be any number of search terms included in the proximity search.

If N is 1, then the search is exactly the same as if the phrase search was used.

For more information on the phrase search, see Phrase Search.

Example:

@ 3 street city @ returns documents that contain the words “street” and “city” not further than 3 words from each other.

Numeric Search

Note: The numeric search functionality is available only starting from the SIETS server version 3.3.

Due to the fact that the SIETS server is indexing not only text information, but it also indexes numeric information, it is possible to perform numeric search.

Numeric search allows searching documents that contain numeric values within a numeric interval.

For example, each document contains information about an object including geographic coordinate information. In that case, the numeric search can be performed to retrieve all objects in definite range of geographic coordinate. Thus, SIETS can be used in online maps, where people can find information on different objects in a definite area.

The numeric search can be performed only together with a text search.

Numeric values in documents are indexed and stored as floating points, no matter if they are integers or floating points in original documents.

Fraction part is stored up to the sixth digits.

To use the numeric search functionality, the search term must be as follows:

- To perform numeric search within a range of two numeric values, enter _textual search term_X .. Y, where X is the minimum value of the search numeric value, and Y is the maximum.
- To perform numeric search for a document that contain numeric value greater than the given, enter _textual search term_>X.
- To perform numeric earch for a document that contain numeric value smaller than the given, enter _textual search term_<X.

It does not matter if textual search term is entered before or after numeric search term.

Example:

Document content:

```
<document>
        <id>32423</id>
        <title>John's profile</title>
        <text>
                <name>John Smith</name>
                <age>32</age>
        </text>
</document>
```

Search query that matches the document:

```
<query>
        <name>Jonh</name> <age>30 .. 40</age>
</query>
<numeric_ordering>center</numeric_ordering>
```

Note: For performing numeric searching for one document tag, as in the previous example the <age> tag, only one numeric interval can be entered. If you enter more than one numeric interval for one tag, then nothing is returned since numeric intervals are joined with the AND logical operation.

For information on performing numeric search for more than tags, see Numeric Search in More Than One Tag.

The <numeric_ordering> tag in the example denotes the order in which search results must be returned.

Possible values for numeric ordering are the following:

Title	Description
none	No numeric ordering is applied.
center	Results that are closer to the mean value of the numeric search interval are returned first. This value is allowed only for numeric search within a range of two numeric values.
ascending	Numeric search results are returned in ascending order.
descending	Numeric search results are returned in descending order.

Numeric Search in More Than One Tag

It is possible to perform numeric search in more than one tag. It means that for each tag that contains numeric information a numeric search range can be performed.

Example:

Document content:

```
<document>
        <id>32423</id>
        <title>John's profile</title>
        <text>
                <name>John Smith</name>
                <age>32</age>
                <children>2</children>
        </text>
</document>
```

Search query that matches the document:

```
<query>
        <name>Jonh</name> <age>30 .. 40</age> <children>&lt;
2</children>
</query>
<numeric_ordering>center</numeric_ordering>
```

Numeric search in more than one tag is especially useful and necessary for geographic coordinate searching, where it is necessary to search for an object by its longitude and latitude.

For numeric search in more than one tag result ordering is combined in one for all tags.

The following table describes result ordering is combined:

Ordering type	Description
ascending	Results are ordered ascending by the sum of all numeric values from tags in which the numeric search is performed.
descending	Results are ordered descending by the sum of all numeric values from tags in which the numeric search is performed.

Ordering type	Description
center	Ordered by shortest distance to the center of intervals in multi-dimensional space where each dimension represents a tag in which the numeric search is performed. Distance to the center of intervals in multi-dimensional space is calculated by the following formula: (x-xc)/xr*(x-xc)/xr + (y-yc)/yr*(y-yc)/yr + …+ (z-zc)/zr*(z-zc)zr, where x, y, z are numeric search intervals xc, yc, zc are centers of each interval, respectively xr, yr, zr are half of numeric interval range, respectively.

Numeric search functionality in several tags or in several dimensions has additional feature that allows returning numeric search results that match:

- a hypercube of all numeric intervals, which is default, or
- only a hypersphere of all numeric intervals.

For example, if geographic coordinates of ATMs in a city are indexed, it is possible to search for an ATM that is not farther than 1 kilometer from a definite location. That is, you need to retrieve only those ATMs that match the circle (a hypersphere with 2 dimensions in this case) with a radius of 1 kilometer.

If in the previous example, the default numeric search is performed, results that match a square with the side length 2 kilometers are returned. This means that also ATMs that are square root of 2, which is approximately 1.41, are returned.

As said before, the default value for the multi dimensional shape feature is a hypercube. Value for the multi dimensional shape feature is defined in the <md_shape> tag, which is included in the siets command syntax.

Possible values for the <md_shape> tag are the following:

Ordering type	Description
cube	Results that match a hypercube are returned.
sphere	Results that match a hypersphere are returned.

Example:

Document content:

```
<document>
        <id>32425</id>
        <title>ATM's profile</title>
        <text>
                <name>ATM</name>
                <x>1.2</x>
                <y>3.7</y>
        </text>
</document>
```

Search query that matches the document and finds ATMs within the distance of 1 kilometer from point (2.0, 4.0):

```
<query>
        <name>ATM</name> <x>1.0 .. 3.0</x> <y>3.0 .. 5.0</y>
</query>
<numeric_ordering>center</numeric_ordering>
<md_shape>sphere</md_shape>
```

Case Sensitivity for Proper Names

It is possible to perform case sensitive search for proper names, which means that case sensitivity is applied for the first letter of a search term.

The case sensitivity feature is switched on or off by setting the <case_sensitive> parameter in the search command's XML request.

For more information on the search command's XML request, see XML Request.

Example:

If the <case_sensitive> parameter is set to YES, and the search query contains "Bank", then the search command returns documents, in which the word "Bank" is with the first capital. Note that in this case, also documents, in which the word "BANK" is with all capitals, are returned, since the case sensitivity is applied only to the first letter of a search term.

Grouping Results by Domain

It is possible to set the maximum number of documents in a search result that are returned form one domain. If this feature is used, in the search result, documents from one domain are grouped together within one result page.

The grouping results by domain feature is defined by setting the <max_from_domain> parameter in the search command's XML request larger than 0.

If the parameter is not set, the default value is 0, which implies that no grouping by domains is performed and no limit is set.

For more information on the search command's XML request, see XML Request.

Filtering Results by Rate

It is possible to filter search results by document rate by setting the minimum and maximum of the rate range within which the rate of a document must be to appear in the search result.

Document rate is of the integer type. However, it is possible to convert any date and time into integer using the UNIX timestamp, which converts a date and time into amount of seconds from 01/01/1970 till the given date and time. Thus, it is possible to set date and time as document rate and to search for document within a certain time interval.

The filtering results by rate feature is defined by setting the <rate_from> and <rate_to> parameters in the search command's XML request.

For more information on the search command's XML request, see XML Request.

Web Friendly Result Navigation

SIETS is designed for use in Web applications in mind. In many cases to display results in Web, the paging functionality is used. The paging functionality implies that

the search result records are divided in parts, where each part is displayed in its own page, and each part contains a fixed amount of records.

The Web friendly result navigation feature is defined by setting the <docs> and <offset> parameters in the search command's XML request.

For more information on the search command's XML request, see XML Request.

Example:

If the <docs> parameter is set to 20, and the <offset> parameter is set to 40, the search command returns results from 40 till 59.

XML Drilldown

XML drilldown is feature that allows grouping documents into hierarchical structure and searching in this structure. Using this feature, you can create catalogues, index files and directories and even much more.

Setting 'classify' policy

Classify policy should be set for those tags for which menus should be generated.

```
<part>
      <location>//document/spectags</location>
      <policy>index=classify</policy>
</part>
```

Policy schema should be set before indexing data.

Document import

Once you have set policy schema you can import documents into storage.

```
<document>
      <id>3049223</id>
      <title>Article</title>
      <text>This is article</text>
      <spectags>
            <type>News_item<type>Comment</type></type>
        <author>John_Smith</author>
      </spectags>
</document>
```

Note that subtags of type or author also can be included to enable multi-level navigation.

Menu generation

Within search requests content tag supply menu tag, which identifies XPath location relative to, the tag for which classify policy is defined of the tag for which to generate menu with hit distribution. Note that menu is generated along with search query and represents number of hits.

```
Example request (single level drilldown):

<siets:command>search</siets:command>
…
<siets:content>
    …
    <query>article</query>
    <menu>/type</menu>
```

```
    ...
</siets:content>
```

```
Example response (single level drilldown)::

<siets:content>
    ...
    <menu>
      <item hits="25">News_item</item>
      <item hits="1">Comment</item>
      <item hits="7">Question</item>
      <item hits="7">Reply</item>
    </menu>
    ...
</siets:content>
```

For multi level drilldown, simply pass correct deeper XPath location. Be sure to add "=<selected value>" to each parent category or you will receive invalid hits.

```
Example resquest (multi level drilldown):

<siets:command>search</siets:command>
...
<siets:content>
    ...
    <query>article</query>
    <menu>/type=News_item/type</menu>
    ...
</siets:content>
```

```
Example response (multi level drilldown):

<siets:content>
    ...
    <menu>
      <item hits="10">Comment</item>
      <item hits="1">Question</item>
      <item hits="3">Reply</item>
    </menu>
    ...
</siets:content>
```

4.5.2.2.HTTP GET Parameters

```
http://host/cgi-bin/siets/api.cgi?command=search&storage=test&query=Jhon
```

4.5.2.3.XML Reply

If the command is executed successfully, the XML reply contains the following command specific data.

```
<siets:content>
    <ignored> common words that are ignored when performing the search </ ignored >
```

```
    <realquery> real query that was used to perform the search, including the derived words
from the wildcard usage and dropped ignored words </ realquery>
    <found> number of documents found </found>
    <hits> approximate total amount of results that match the search query </hits>
    <more> number that indicates how many more documents that match the search query
are found, but are not returned to the result set yet, a precise number if in the form of =N,
and an at least number if in the form of >N</more>
    <from> documents in the result set  within a numerical range: the FROM value </from>
    <to> documents in the result set  within a numerical range: the TO value </to>
    <results>
        <document> meta data of the document found </document> <!-- This element is
repeated for all documents found. -->
    </results>
</siets:content>
```

If the command is not executed successfully, an error is returned. For more information on errors, see Error Handling.

4.5.3.Select

Note: The select command is available only starting from the SIETS server version 3.2.6.

The select command searches for document by their identifiers. It is possible to select one document by a precisely entered document identifier or to use wildcard pattern to select all documents that identifiers match the wildcard pattern entered. For example, if only the asterisk * is entered, identifiers for all document in the SIETS storage will be returned.

The default number of document identifiers returned to result set is 1024, but this number can be changed by entering a different number in the <docs> tag.

4.5.3.1.XML Request

```
<siets:content>
    <document>
        <id>document id *</id>
    </document>
    <docs> number of document identifiers in the result set </docs>
    <offset> intend from the beginning of the result set</offset>
</siets:content>
```

4.5.3.2.XML Reply

If the command is executed successfully, the XML reply contains the following command specific data.

```
<siets:content>
    <found> number of document identifiers matched </found>
    <from> document identifiers in the result set  within a numerical range: the FROM value
</from>
    <to> document identifiers in the result set  within a numerical range: the TO value </to>
    <results>
        <id> meta data of the document found </id> <!-- This element is repeated for all
document identifiers matched. -->
    </results>
</siets:content>
```

If the command is not executed successfully, an error is returned. For more information on errors, see Error Handling.

4.5.4.Similar

The similar command searches for similar documents in the SIETS storage to a textual information, which is given directly, or which is contained by a document. The textual information, to which similar documents are searched for, is also referred as the input text.

The algorithm that is searching for similar documents uses statistical information about the number of times words contained by the input text, or so called keywords, appear in documents and finds similar documents to the input text fragment or document with a given ID.

You must take into account that the algorithm uses statistical information about words and does not know their meaning. Therefore, similar documents might not be semantically alike, however, praxis, when working with large text collections that contain medium large documents, shows that the algorithm works fine.

4.5.4.1.XML Request

```
<siets:content>
    <id> document id to which similar documents must be searched for ** </id>
    <text> textual information to which similar documents must be searched for ** </text>
    <len> number of keywords in the input text * </len>
    <quota> minimal amount of keywords that must be found in documents, which are
returned the search result *</quota>
    <docs> number of documents to be retuned in the result set </docs>
    <offset> intend from the beginning of the result set </offset>
</siets:content>
```

For large text collections in the SIETS storage, praxis shows that the len element equal to 20 and the quota element equal to 4 gives the best results. However, you can experiment to find the best values for your specific text collection.

The two asterisks ** means that only one from the two elements must be entered, in other means, the relationship between these two elements is XOR.

4.5.4.2.HTTP GET Parameters

```
http://host/cgi-
bin/siets/api.cgi?command=similar&storage=test&id=Doc1&len=20&quota=4
http://host/cgi-
bin/siets/api.cgi?command=similar&storage=test&text=Jhon&len=20&quota=4
```

4.5.4.3.XML Reply

If the command is executed successfully, the XML reply contains the following command specific data.

```
<siets:content>
    <found> number of documents found </found>
    <hits> approximate total amount of results that match the search query </hits>
    <more> number that indicates, how many more documents that match the search query
are found, but are not returned to the result set yet, a precise number if in the form of =N,
and the minimum number if in the form of >N</more>
    <from> documents in the result set  within a numerical range: the FROM value </from>
    <to> documents in the result set  within a numerical range: the TO value </to>
    <results>
        <document> meta data of the document found </document> <!-- This element is
repeated for all documents found. -->
```

```
    </results>
</siets:content>
```

If the command is not executed successfully, an error is returned. For more information on errors, see Error Handling.

4.5.5.Alternatives

If the alternatives search is performed, the system returns a set of alternative words from the SIETS storage vocabulary, which are similar in spelling or has a different language declination, for example, if you enter "bote", then "bite" and "byte" are offered for searching. Note that only words from the SIETS storage are returned.

This feature can be used for fuzzy searches and for spelling error corrections.

Alternative words are returned from the vocabulary, which ensures that the alternative words are actual words that are in imported to the SIETS storage. When searching alternative words, the alternatives command considers the statistical information about the occurrence of the alternative word in the vocabulary, and the similarity of the alternative word to the search term. In other words, alternatives that occur in the SIETS storage more often and that are more similar to the search term are returned.

4.5.5.1.XML Request

```
<siets:content>
    <query> search query * </siets_query>
    <cr> Minimum ratio to include the alternative in the search query between the occurrence of the alternative and the occurrence of the search term. If you increase this parameter, there are less number of results returned to the result set, however performance is improved.</cr><!-- Functionality of this tag is available only starting from the SIETS server version 3.2.8.-->
    <idif> Maximum number that indicates how much does the alternative differs from the search term, the greater the idif value, the greater the difference. If you increase this parameter, there are greater number of results returned to the result set, however performance is reduced.</idif><!-- Functionality of this tag is available only starting from the SIETS server version 3.2.8.-->
    <h> Minimum number that gives an overall estimation of the quality of the alternative, the greater the cr value and the smaller the idif value, the grater the h value. If you increase this parameter, there are less number of results returned to the result set, however performance is improved.<h><!-- Functionality of this tag is available only starting from the SIETS server version 3.2.8.-->
</siets:content>
```

If values for the cr, idif, or h tags are not defined, corresponding parameters set in the SIETS storage configuration file are used.

For more information configuring SIETS storage, see the *SIETS Administration and Configuration Guide*.

4.5.5.2.HTTP GET Parameters

```
http://host/cgi-bin/siets/api.cgi?command=alternatives&storage=test&query=Jhon
```

4.5.5.3.XML Reply

If the command is executed successfully, the XML reply contains the following command specific data.

```
<siets:content>
```

```
    <alternatives_list>
        <alternatives>
                <to> alternative search term </to>
                <count> number of times the alternative search term occurs in the SIETS
storage</count>
                <word count="number of times the alternative occurs in the SIETS storage"
cr="ratio between the occurrence of the alternative and the occurrence of the search term"
idif="number that indicates how much does the alternative differs from the search term, the
greater the idif value, the greater the difference" h="number that gives an overall estimation
of the quality of the alternative, the greater the cr value and the smaller the idif value, the
grater the h value"> alternative </word><!-- This element is repeated for each alternative
word.-->
        </alternatives><!-- This element is repeated for each search term.-->
    </alternatives_list>
</siets:content>
```

If the command is not executed successfully, an error is returned. For more information on errors, see Error Handling.

4.5.6.List-last

Note: The list-last command is available only starting from the SIETS server version 3.2.9.

The list-last command searched for documents in the SIETS storage that most recently have been inserted, updated, or replaced, using the insert, update, or replace commands, respectively.

4.5.6.1.XML Request

```
<siets:content>
    <docs> number of documents in the result set </docs>
    <offset> intend from the beginning of the result set</offset>
</siets:content>
```

4.5.6.2.HTTP GET Parameters

```
http://host/cgi-bin/siets/api.cgi?command=list-last&storage=test&docs=10&offset=100
```

4.5.6.3.XML Reply

If the command is executed successfully, the XML reply contains the following command specific data.

```
<siets:content>
    <found>number of documents returned to the result set</found>
    <results>
        <document> meta data for the list-last command</document><!-- This element is
repeated for all documents found. -->
    </results>
</siets:content>
```

Meta data for the list-last command is information included in tags, for which the policy list is set to YES. By default, these are id, title, and rate tags.

For more information on policies, see Importing XML Structured Data.

If the command is not executed successfully, an error is returned. For more information on errors, see Error Handling.

4.6.Alerts

SIETS Server allows user to use alerting functionality. Alerts are defined as search queries that can be performed against storage inside server. Alerts are not triggered automatically; special command must be used. This is done, to give user application even more flexibility in alert handling.

Alerting API commands are sent to server using standard SIETS XML messaging.

4.6.1.Adding trigger

This command add trigger identified with supplied ID that will match documents against query supplied in filter tag.

```
<siets:command>add_trigger</siets:command>
<siets:content>
    <id>Trigger id</id>
    <filter>Trigger filter query</filter>
    <recipient>Recipient of notification</recipient>
</siets:content>
```

4.6.2.Removing trigger

This command removes specific trigger.

```
<siets:command>remove_trigger</siets:command>
<siets:content>
    <id>Trigger id</id>
</siets:content>
```

4.6.3.Clear triggers

This commnad clears all triggers.

```
<siets:command>clear_triggers</siets:command>
```

4.6.4.Examining document against triggers

This command test document thats ID is supplied against all triggers. If notify parameter is set to yes shell script is executed for each trigger that matches document. Also in reply to this command list of trigger-id's that matched document is returned.

```
<siets:command>examine</siets:command>
<siets:content>
    <document>
        <id>document id to examine</id>
    </document>
    <notify>yes/no – to send message or not</notify>
</siets:content>
```

4.6.5.Configuration parameters

Storage configuration can be used, to specify shell script that will be executed, when trigger is matched against document.

```
<config>
    <alerts>
        <action>Shell script to execute</action>
```

```
    </alerts>
</config>
```

4.7.Error Handling

If a command sent to the SIETS server is not executed successfully, an error is returned in the following XML reply message:

```
<?xml version="1.0"?>
<siets:reply>
    <siets:timestamp> date and time </siets:timestamp>
    <siets:storage> storage name </siets:storage>
    <siets:requestid>XML request ID</siets:requestid>
    <siets:error>
        <code>error code</code>
        <text> error textual message</text>
        <level> error severity</level>
        <source>subsystem in which the error occurred</source>
    </siets:error>
    <siets:seconds> time period in which the XML reply is returned </siets:seconds>
</siets:reply>
```

The error severity can be one of the following:

Title	Description
Warning	Returned when the command is executed successfully, but there are some problem indications
Failed	Returned when incorrect input data.
Error	Returned when error in the command execution.
Fatal	Returned when the system is not functioning.

The purpose of the error severity is to inform the system:

- If the error severity is fatal or error, the system work must be interrupted and the system administrator must be informed.
- If the error severity is failed or warning, the errors can be logged and analyzed, while the system work can be continued.

SIETS is a transaction-based system, which means that commands has a predefined timeout period. If a command is not executed during this predefined timeout period, the command returns the error.

It is possible to define a timeout period for the request, or configure it for the SIETS server.

For more information on configuring timeout periods for the SIETS server, see the *SIETS Administration and Configuration Guide*.

5. SIETS CLUSTERING

This section describes SIETS clustering technology and provides general steps for working with it. Using the SIETS clustering technology, several SIETS servers can be joined into one cluster, which enables that search can be performed in a text collection of an unlimited size.

This section contains the following topics:

- Principles
- Creating SIETS Cluster

5.1.Principles

A SIETS cluster consists of nodes. Each node is a computer, on which the SIETS server is installed.

The SIETS cluster technology has a transparent architecture, which implies that each node is fully functional on its own.

5.2.Creating SIETS Cluster

To create a SIETS cluster, proceed as follows:

1. Let us assume that we have a very large text collection that is too big to be stored and worked with on single computer. In that case, estimate in how many equal in size parts the text collection can be divided so that each part can be stored and worked with on single computer.
2. On the number of computers estimated in the previous step, install the SIETS server on each of them.

   For more information on installing SIETS servers, see the *SIETS Installation Guide*.
3. On each SIETS server installed in the previous step, create a SIETS storage with the same name for all nodes, that is SIETS servers.

   For more information on creating storages on SIETS servers, see the *SIETS Administration and Configuration Guide*.
4. Create an application, which imports each part of the text collection to its own node.

Note: If you want to use the `domain` element for grouping search results, then all documents of the same domain must be imported to the same node. Otherwise, that is, if documents of the same domain are stored on several nodes, when performing a search in the whole text collection results will not be grouped by domains properly.

When you have created a SIETS cluster, you can either:

- Connect to only one SIETS cluster's node and, thus, perform a search only within a part of the whole text collection.
- Connect to any of the SIETS cluster's nodes with the parameter `cluster=yes`. The node connects to all other nodes in the SIETS cluster and thus a unified result of a search query in the whole text collection is created.

6. SIETS CONSOLE

This section describes the SIETS console and related principles that must be understood before working with it, and contains step-by-step instructions for running the SIETS console.

This section contains the following topics:

- Overview
- Parameters
- SIETS Console Parameters
- Templates
- Built-in Commands
- Running SIETS Console

6.1.Overview

SIETS console is a simple text application for accessing the SIETS server directly using SIETS API.

The following steps provide a general description of how the SIETS server is accessed using the SIETS console:

1. Users enter command prompt commands for the SIETS server directly through the SIETS console, which is run from the.
2. The SIETS console submits each API command to the SIETS server.
3. The SIETS server responds by returning XML replies to the SIETS console, which are dumped on the command prompt screen.

Generally, in the SIETS console, the API command is entered as the command name followed by it's parameters in predefined order, and the reply is returned as XML replies described in SIETS API Specification.

Command's parameters correspond to tag names in API XML requests and names of HTTP GET/POST parameters.

6.2.Parameters

Each SIETS API command parameter has a name. Some command parameters have default values, and some do not. When executing a command, if the parameter value is not set, the SIETS console takes the default value, or, if the parameter does not have the default value, it asks for the value interactively by displaying the parameter name and the equal sign =.

6.3.SIETS Console Parameters

In the SIETS console you do not need to run the session initiation function. The session is initiated already when you run the SIETS console. However, you have to set the SIETS console parameters, which are the same as for the session initiation function plus the encoding parameter:

Title	Description
host	SIETS server IP address or a host name.
storage	SIETS storage name.
user	User name.
password	User password.
timeout	Function timeout period. If the function is not executed during this predefined timeout period, the command returns the error.
encoding	Character encoding of input data and reply. **Note:** The textual data must comply with the encoding defined in a function parameter, or else the system returns a parsing error. **Note:** The terminal that you are using to access the SIETS console must be configured for the encoding you have set, and it must be capable to display all characters of the encoding, or else, some characters may not be displayed correctly.

If these parameters are not set when starting the SIETS console, the SIETS console asks for them interactively by displaying the parameter name and the equal sign = when executing the first function.

When calling the SIETS console, these parameters are set as options, for example, -u. When already working in the SIETS console, these parameters are set as parameter names, for example, user.

The SIETS console parameters can be changed in any time when running the SIETS console.

6.4.Templates

For each function there is a template that can be viewed before executing the function to ensure that all parameters are set. A function template contains the syntax of the SIETS API command and its parameters, and the corresponding XML request syntax of the corresponding command.

6.5.Built-in Commands

There are the following built-in commands in the SIETS console:

Title	Parameter	Description
help		Displays the list of built-in commands and their descriptions and the list of templates.
echo	<SIETS console parameter name>	Prints the SIETS console parameter value.
show	<function name>	Prints the template of the function.
exit		Exits the SIETS console.

6.6.Running SIETS Console

To run the SIETS console, proceed as follows:

1. Login to the computer directly, using Telnet, or secure shell, where the SIETS server is installed.
2. To start the SIETS application, enter the following command with the options, which each corresponds to the SIETS console parameter:

   sietsco -h <SIETS server IP address or a host name >, -p <SIETS server port number>, -s <SIETS storage name>, -u <user name>, -w <user password>

   Note: You can skip the options, however, if the SIETS console parameters are not set when starting the SIETS console, the SIETS console asks for them interactively by displaying the parameter name and the equal sign = when executing the first function.

3. Start executing SIETS API commands. To view the function syntax, run the built-in command show described in Built-in Commands.

   There are the following two syntaxes for executing the SIETS API commands from the SIETS console:

   - function (param1, param2, … paramN)
   - function param1 param2 … paramN

   All parameters must be set in the order they are presented in the function template. If you want to use the default value for the parameter, you can leave it empty. If the parameter you want to have the default value is at the end of the parameters list, you can simply drop it. However, if the parameter you want to have the default value is at the beginning or in the middle of the parameters list, to leave it empty, you must use the function executing syntax with the brackets and parameters separated with commas.

   Example:

   search (john,50,,yes) sets the parameter values to the following: query=john, docs=50, offset=10, relevance=yes, rate_form=0, and rate_to=0

   search john 50 sets the parameter values to the following: query=john, docs=50, offset=10, relevance=no, rate_form=0, and rate_to=0

7. MESSAGE FILES

This section contains the following topics:

- Overview
- Structure of Message Files
- Running siets-load

7.1.Overview

Most common way of working with SIETS is sending and receiving XML requests and replies via HTTP API, which each contains a request and a reply of one SIETS command, respectively. However, there might be cases when it is more convenient to send a set of XML requests containing many SIETS commands for modifying contents of the SIETS storage directly to the SIETS server. For that reason the message files functionality has been included in the SIETS system.

The message files functionality implies that unlimited amount of XML requests can be stored in one or several files called message files and these message files can be uploaded to the SIETS server for execution using the siets-load command from the command promt.

The message files functionality can be used only from the same computer on which the SIETS server is installed.

7.2.Structure of Message Files

Message files have .msg extension and the following structure:

```
<msg code="0" len="N"> data N bytes of XML message </msg>
```

Each XML request must be enclosed in the mgs tags.

The code parameter always must be set to 0.

The len parameter denotes number of bytes an XML request text contains. It is also possible to set the len parameter to -1, in that case, the closing tag </msg> of the XML request is searched when performing the siets-load command.

Note: If a XML request contains a string </msg>, then the len parameter must not be set to -1, since in that case it would be considered as the message closing tag.

First character of XML request enclosed in the msg tags must follow immediately after the opening msg tag. There cannot be any space symbol or new line symbol between the last character of the opening msg tag and the first character of the XML request.

The following is an example of a message file:

```
<msg code="0" len="-1"><?xml version="1.0" encoding="US-ASCII"?>
<siets:request xmlns:siets="www.siets.net">
            <siets:storage>My storage</siets:storage>
            <siets:command>insert</siets:command>
            <siets:requestid>101</siets:requestid>
            <siets:appilcation>sample application</siets:application>
            <siets:user>john_smith</siets:user>
        <siets:passwd>qwerty</siets:passwd>
```

```
            <siets:content>
                <document>
                        <id> 56 </id>
                        <title> My document </title>
                        <rate> 10 </rate>
                        <text> The quick brown fox jumps over the lazy dog. </text>
                </document>
            </siets:content>
</siets_request>
</msg>
<msg code="0" len="356"><?xml version="1.0" encoding="US-ASCII"?>
<siets:request xmlns:siets="www.siets.net">
            <siets:storage>My storage</siets:storage>
            <siets:command>index</siets:command>
            <siets:requestid>102</siets:requestid>
            <siets:appilcation>sample application</siets:application>
            <siets:user>john_smith</siets:user>
        <siets:passwd>qwerty</siets:passwd>
</siets_request>
</msg>
```

7.3.Running siets-load

To run the siets-load command, proceed as follows:

1. In the command promt, open the directory containing the message files you want to run.
2. Run the siets-load executable file with the following parameters:

Parameter	Description
-d	Full path of the SIETS storage directory. By default, SIETS storage directories are located in /usr/local/siets/data/.
-o	If set, denotes if the siets-load command is performed once on message files of the current directory. If not set, the directory is being scanned all time and, each time a message file appears in it, the siets-load command is performed.

By default, the siets-load executable file is installed in /usr/local/siets/bin/.

Message files are being loaded to the SIETS server and renamed from *.msg to *.msg.ex, so that the siets-load command does not load the same message file once again.

8. USE CASES

This section contains sample applications or use cases written for the SIETS system. Each use case is in a separate section and contains a short description and source code.

This section contains the following use cases:

- Use Case in C: Importing Text Files
- Use Case in Perl: Importing Text Files
- Use Case in PHP: Searching SIETS Storage and Returning Results in HTML
- Use Case in ASP: Searching SIETS Storage and Returning Results in HTML
- Use Case in Java: Searching SIETS Storage from applet
- Use Case in FoxPro I: Updating SIETS Storage from Database Table Data
- Use Case in FoxPro II: Searching SIETS Storage and Returning Results to Cursor

8.1.Use Case in C: Importing Text Files

This application is implemented in the C programming language.

The application reads files from the file system and imports them to the SIETS storage. The application is receiving file names as command line arguments.

It also detects whether the file is a text file or a binary file by counting whitespaces in them: if a file contains relatively less whitespaces, it is considered to be a binary file, and if a file contains relatively more whitespaces, it is considered to be a text file.

```
/**
 * This application is implemented in the C programming language.
 *
 * The application reads files from the file system and imports them to the
 * SIETS storage via HTTP POST interface using libcurl.
 *
 * The application receives file names as command line arguments.
 *
 * It also detects whether the file is a text file or a binary file by counting
 * whitespaces in it: if a file contains relatively less whitespaces, it is
 * considered to be a binary file, and if a file contains relatively more
 * whitespaces, it is considered to be a text file.
 */

/* include standard headers */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <ctype.h>

/* libcurl */

#include <curl/curl.h>

/* connection parameters */
```

```
char *url = "http://195.244.157.173/cgi-bin/siets/api.cgi";
char *storage = "ljur";
char *user = "guest";
char *passwd = "guest";
char *encoding = "US-ASCII";
char *post_fmt =
"storage=%s&command=insert&user=%s&password=%s&id=%s&title=%s&rate=%d&text=%s&
encoding=%s";

#define REQUIRED_WHITESPACE_FRACTION 0.12

typedef struct { int len, used; char *buf; } curl_reply;



size_t read_reply(void *buffer, size_t size, size_t nmemb, void *userp)
{
    int new_len;

    curl_reply *r = (curl_reply *) userp;
    for (new_len = r->len; new_len < r->used + size * nmemb + 1; new_len *= 2);
    if (new_len > r->len) r->buf = realloc(r->buf, new_len);
    memcpy(r->buf + r->used, buffer, size * nmemb);
    r->len = new_len;
    r->used += size * nmemb;
    r->buf[r->used] = '\0';

    return size * nmemb;
}



int main(int argc, char *argv[])
{
    CURL *curl_handle;
    char *storage_esc, *user_esc, *passwd_esc, *title_esc, *text_esc, *encoding_esc;
    curl_reply reply;
    char *err_buf[CURL_ERROR_SIZE];
    int i;

    if (argc == 1) {
        printf("Usage: [-r url] [-s storage] [-u user] [-p password] [-e encoding] files\n");
        return 0;
    }

    /* read options */

    for (i = 1; i < argc; i++) {
        if (argv[i][0] == '-') {
                if (i + 1 >= argc) break; /* no option value */
                switch(argv[i][1]) {
                case 'r':
                        url = argv[i+1];
                        break;
                case 's':
                        storage = argv[i+1];
                        break;
                case 'u':
```

```
                        user = argv[i+1];
                        break;
                case 'p':
                        passwd = argv[i+1];
                        break;
                case 'e':
                        encoding = argv[i+1];
                        break;
                default:
                        printf("Unknown option: %s\n", argv[i]);
                        break;
                }
                i++;
        }
}

/* initialization */

curl_global_init(CURL_GLOBAL_ALL);
curl_handle = curl_easy_init();
curl_easy_setopt(curl_handle, CURLOPT_URL, url);
curl_easy_setopt(curl_handle, CURLOPT_WRITEFUNCTION, read_reply);
curl_easy_setopt(curl_handle, CURLOPT_WRITEDATA, &reply);
curl_easy_setopt(curl_handle, CURLOPT_ERRORBUFFER, err_buf);

storage_esc = curl_escape(storage, 0);
user_esc = curl_escape(user, 0);
passwd_esc = curl_escape(passwd, 0);
encoding_esc = curl_escape(encoding, 0);

/* names of files to be imported are passed as arguments
 * process each of them */

for (i = 1; i < argc; i++) {
    FILE *f;
    struct stat st;
    int k, spaces;
    char *buf, *post_data;

    /* check if argument is option */

    if (argv[i][0] == '-') {
            if (i + 1 >= argc) break; /* no option value */
            i++;
            continue;
    }

    printf("Reading file: '%s'\n", argv[i]);

    /* open file */

    f = fopen(argv[i], "r");
    if (!f) {
            fprintf(stderr, "Couldn't open file '%s'\n", argv[i]);
            continue;
    }

    /* retrieve file information */
```

```
        if (fstat(fileno(f), &st) < 0) {
                fprintf(stderr, "Filesystem error retrieving info on '%s'\n", argv[i]);
                fclose(f);
                continue;
        }
        if (!S_ISREG(st.st_mode)) {
                fprintf(stderr, "File '%s' is not regular file\n", argv[i]);
                fclose(f);
                continue;
        }
        printf("\tSize: %d bytes\n", st.st_size);

        /* read all of it into memory */

        /* note: this sample program asumes all of file fits into memory
         * so if you need to work with larger files figure out something else */

        buf = (char *) malloc(st.st_size + 1);
        if (!buf) {
                fprintf(stderr, "Memory allocation failed\n");
        }
        k = fread(buf, 1, st.st_size, f);
        fclose(f);
        if (k < st.st_size) {
                fprintf(stderr, "Error reading file\n");
                free(buf);
                continue;
        }
        buf[k] = '\0';

        /* see if it is text file
         * estimate that by counting white space:
         * natural language text in contrary to binary data
         * must contain significant portion of whitespace */

        spaces = 0;
        for (k = 0; k < st.st_size; k++) {
                if (isspace(buf[k])) spaces++;
        }
        if (spaces < st.st_size * REQUIRED_WHITESPACE_FRACTION) {
                printf("\tBinary file: ignored\n");
                free(buf);
                continue;
        }

        /* execute SIETS insert command through HTTP POST interface */

        title_esc = curl_escape(argv[i], 0);
        text_esc = curl_escape(buf, k);

        post_data = malloc(strlen(storage_esc) + strlen(user_esc) + strlen(passwd_esc) +
strlen(post_fmt) + 2 * strlen(title_esc) + 20 + strlen(text_esc) + strlen(encoding_esc));
        sprintf(post_data, post_fmt, storage_esc, user_esc, passwd_esc, title_esc, title_esc, 100,
text_esc, encoding_esc);
        curl_easy_setopt(curl_handle, CURLOPT_POSTFIELDS, post_data);

        reply.buf = malloc(reply.len = 1);
        reply.used = 0;
        if (curl_easy_perform(curl_handle) != CURLE_OK) {
```

```
                fprintf(stderr, "Error connecting to SIETS server: %s\n", err_buf);
        } else if (strstr(reply.buf, "<siets:error>")) { /* simplified error check */
                *((char *) strstr(reply.buf, "</text>")) = '\0';
                fprintf(stderr, "Error returned from SIETS server: %s\n", strstr(reply.buf,
"<text>") + 6);
        } else {
                *((char *) strstr(reply.buf, "</docid>")) = '\0';
                printf("Document inserted with id %s\n", strstr(reply.buf, "<docid>") + 7);
        }

        /* cleanup */

        free(buf);
        free(reply.buf);
        free(title_esc);
        free(text_esc);
        free(post_data);
    }

    /* final cleanup */

    free(storage_esc);
    free(user_esc);
    free(passwd_esc);
    free(encoding_esc);

    curl_easy_cleanup(curl_handle);
    curl_global_cleanup();

    return 0;
}
```

8.2.Use Case in Perl: Importing Text Files

This application is implemented in the Perl programming language.

The application reads files from the file system and imports them to the SIETS storage. The application is receiving file names as command line arguments.

```
#
# This application is implemented in the Perl programming language.
#
# The application reads files from the file system and imports them to the
# SIETS storage through HTTP POST interface using libcurl.
#
# The application receives file names as command line arguments.
#
# It also detects whether the file is a text file or a binary file by counting
# whitespaces in it: if a file contains relatively less whitespaces, it is
# considered to be a binary file, and if a file contains relatively more
# whitespaces, it is considered to be a text file.
#

use HTTP::Request::Common;
use LWP::UserAgent;
use File::stat;

# connection parameters
```

```
$url = "http://127.0.0.1/cgi-bin/siets/api.cgi";
$storage = "test";
$user = "guest";
$passwd = "guest";
$encoding = "US-ASCII";

$REQUIRED_WHITESPACE_FRACTION = 0.12;


if (@ARGV == 0) {
    print "Usage: [-r url] [-s storage] [-u user] [-p password] [-e encoding] files\n";
    exit;
}

# read options
for ($i = 0; $i < @ARGV; $i++) {
    if (substr($ARGV[$i], 0, 1) eq '-') {
        if ($i + 1 >= @ARGV) { last; } # no option value
        $opt = substr($ARGV[$i], 1, 1);
        $val = $ARGV[$i + 1];
        if ($opt eq 'r') {
                $url = $val;
        } elsif ($opt eq 's') {
                $storage = $val;
        } elsif ($opt eq 'u') {
                $user = $val;
        } elsif ($opt eq 'p') {
                $passwd = $val;
        } elsif ($opt eq 'e') {
                $encoding = $val;
        } else {
                print "Unknown option: ", $ARGV[$i], "\n";
        }
        $i++;
    }
}

$ua = LWP::UserAgent->new;

# names of files to be imported are passed as arguments
# process each of them
for ($i = 0; $i < @ARGV; $i++) {

    # check if argument is option
    if (substr($ARGV[$i], 0, 1) eq '-') {
        if ($i + 1 >= @ARGV) { last; } # no option value
        $i++;
        next;
    }

    print "Reading file: '", $fn = $ARGV[$i], "'\n";

    # open file
    if (open(f, $fn)) {

        # retrieve file information
        if ($st = stat(*f)) {
                if (($st->mode & S_IFMT) == S_IFREG) {
                        print "\tSize: ", $st->size, " bytes\n";
```

```
                        # read all of it into memory
                        # note: this sample program asumes all of file fits into memory
                        # so if you need to work with larger files figure out something else
                        if (sysread(*f, $buf, $st->size) == $st->size) {

                                # see if it is text file
                                # estimate that by counting whitespace in it:
                                # natural language text in contrary to binary data
                                # must contain significant portion of whitespace
                                $nspaces = $buf =~ s/(\s)/$1/g;
                                if ($nspaces >= $st->size *
$REQUIRED_WHITESPACE_FRACTION) {

                                        # execute SIETS insert command through HTTP POST
interface */
                                        $response = $ua->request(POST $url, [
                                                storage => $storage,
                                                command => 'insert',
                                                user => $user,
                                                password => $passwd,
                                                id => $fn,
                                                title => $fn,
                                                rate => 100,
                                                text => $buf,
                                                encoding => $encoding
                                        ]);

                                        if ($response->is_success && $response->content) {
                                                if ($response->content !~ /<siets:error>/) { #
simplified error check
                                                        $response->content =~
/<docid>([^<]*)<\/docid>/;
                                                        print "Document inserted: docid = $1\n";
                                                } else {
                                                        $response->content =~
/<code>([^<]*)<\/code>/;
                                                        print STDERR "Error returned from SIETS
server: $1 - ";
                                                        $response->content =~
/<text>([^<]*)<\/text>/;
                                                        print STDERR "$1\n";
                                                }
                                        } else {
                                                print STDERR "Error connecting to SIETS server:
", $response->code, ' - ', $response->message, "\n";
                                        }
                                } else {
                                        print "\tBinary file: ignored\n";
                                }
                        } else {
                                print STDERR "Error reading file\n";
                        }
                } else {
                        print STDERR "File '$fn' is not a regular file\n";
                }
        } else {
                print STDERR "Filesystem error retrieving info on '$fn'\n";
        }
```

```
        close(f);
    } else {
        print STDERR "Could not open file '$fn'\n";
    }
}
```

8.3.Use Case in PHP: Searching SIETS Storage and Returning Results in HTML

This application is implemented in the PHP programming language.

The application searches the SIETS storage and returns the results in HTML.

```
<?

//
// This application is implemented in the PHP programming language.
//
// The application searches the SIETS storage using HTTP API and returns the
// results in HTML.
//

$SIETS_SERVER = "http://195.244.157.173/cgi-bin/siets/api.cgi";
$SIETS_STORAGE = "ljur";
$SIETS_USER = "guest";
$SIETS_PASSWD = "guest";

//search query
$query = $_GET["q"];

//current position in results
$curr_position = $_GET["p"];

//data encoding
$encoding = "UTF-8";

//send http header with correct encoding
send_headers($encoding);

//max results on page
$result_on_page = 10;

if (empty($curr_position) || $curr_position < 0) {
    $curr_position = 0;
}

//max page from one domain to show
$max_page_from_domain = 2;

$xml_text = file_get_contents($SIETS_SERVER .
"?storage=$SIETS_STORAGE&command=search&user=" . urlencode($SIETS_USER) .
"&password=" . urlencode($SIETS_PASSWD) . "&query=" . urlencode($query) .
"&docs=$result_on_page&offset=$curr_position&from_domain=$max_page_from_domain&encod
ing=UTF-8");
if ($xml_text == "") {
    die("Siets_search error!");
}
```

```
//initialize xml to array object
$xml2a = new XMLToArray();

//parse xml
$root_node = $xml2a->parse($xml_text);

//pop root node from array
$siets_reply = array_shift($root_node["_ELEMENTS"]);

//array for storing data from search results
//like total time spent, hits, and so on
$spec_data = array();

// examining SIETS reply elements
foreach ($siets_reply["_ELEMENTS"] as $siets_reply_el) {
    if ($siets_reply_el["_NAME"] == "seconds") {
    $spec_data[$siets_reply_el["_NAME"]] = $siets_reply_el["_DATA"];
}

// examining SIETS content elements folder
foreach ($siets_reply_el["_ELEMENTS"] as $siets_content) {
    $spec_data[$siets_content["_NAME"]] = $siets_content["_DATA"];
    $last_domain = '';
    foreach($siets_content["_ELEMENTS"] as $results) {
        $tit = "";
        $others = "";
        //parse each document tag from the result set
        foreach($results["_ELEMENTS"] as $documents) {
                switch ($documents["_NAME"]) {
                case "title" :
                        $tit_array = explode(":_:_:", $documents["_DATA"]);
                        $tit .= '<b>'.$tit_array[0].'</b>';
                        break;
                case "link" :
                        $others .= '<br/><font size="-1"> Link:
'.$documents["_DATA"].'</font>';
                        break;
                case "domain" :
                        if ($last_domain == $documents["_DATA"])
                                $blockquote = TRUE;
                        else
                                $blockquote = FALSE;
                        $others .= '<br/><font size="-1"> Domain:
<i>'.$documents["_DATA"].'</i></font>';
                        $last_domain = $documents["_DATA"];
                        break;
                case "rate" :
                        break;
                case "info" :
                        break;
                case "text" :
                        if (!empty($documents["_DATA"]))
                                $others .= '<br/><font size="-1"> Snippet:
<i>'.$documents["_DATA"].'</i></font>';
                                break;
                        }
                }
```

```
                //tab domains
                if ($blockquote)
                        $output .= '<blockquote>'.$tit.$others.'</blockquote>';
                else
                        $output .= '<br>'.$tit.$others.'<br>';
        }
    }
}

if ($spec_data["hits"] == 0) {
    $output = "Your search <b>$query</b> did not match any documents!";
} else {
    //page listing
    $from = $curr_position + 1;
    $to = $curr_position + strval($spec_data["found"]);
    $list_begin_pos=0;
    $list_end_pos=$curr_position+($result_on_page*10);
    $page_list .= "<font size=\"-1\">";
    $p = 1;

    if($curr_position > ($result_on_page * 10)){
        $list_begin_pos=$curr_position-($result_on_page*10);
        $p=intval($list_begin_pos/$result_on_page)+1;
    }

    if ($curr_position > 0) {
        $page_list .= " <a href=\"sietsu.php?p=". ($curr_position - $result_on_page)
."&q=".urlencode(stripslashes($query))."\">&lt;&lt;Previous</a>   ";
    }

    $more_tag = $spec_data["more"];
    if ($more_tag[0] == '=') {
        $more = substr($more_tag,1);
    } else {
        $more = substr($more_tag,1) + 1;
    }

    for ($i = $list_begin_pos; $i - ($curr_position + $more) < $result_on_page && $i <
$list_end_pos; $i+= $result_on_page) {
        if($i>=1000) break;
        if ($i != $curr_position) {
                $page_list .= "<a
href=\"sietsu.php?p=$i&q=".urlencode(stripslashes($query)).(!empty($dir)?"&dir=$dir":"")."\">$
p</a> ";
        } else {
                $page_list .= "<b>$p </b>";
        }
        $p++;
    }

    if (($result_on_page+$curr_position)-($curr_position+$more) < 10 && $curr_position +
$result_on_page < 1000) {
        $page_list .= "   <a href=\"sietsu.php?p=".($curr_position +
$result_on_page)."&q=".urlencode(stripslashes($query))."\">Next&gt;&gt;</a>";
    }

    $page_list .= "</font>";
    //end of page listing
}
```

```
//echo output to client
echo '
<html>
<head>
<style><!--
body,td,p,a{font-family:arial,sans-serif;}
.servkat{color:003399; text-decoration:none}
.homepage{color:003399; text-decoration:none; font-size:10px;}
//-->
</style>
</head>
<body>
<table>
<tr bgcolor=\"#cccc66\">
<td><font size=\"-1\">  Searched for: <b>'.$query.'</b> Results: <b>'.$from.'</b> -
<b>'.$to.'</b> from <b>'.$spec_data["hits"].'</b> Search lasted
<b>'.$spec_data["seconds"].'</b> seconds </font> </td>
<tr>
</table>
'.$output.'<br>
'.$page_list.'
</body>
</html>';

//############################################################

class XMLToArray
{
    //---------------------------------------------------------------------

    // private variables

    var $parser;
    var $node_stack = array();

    //---------------------------------------------------------------------

    // PUBLIC
    // If a string is passed in, parse it right away.

    function XMLToArray($xmlstring="")
    {
        if ($xmlstring) return($this->parse($xmlstring));
        return(true);
    }

    //---------------------------------------------------------------------

    // PUBLIC
    // Parse a text string containing valid XML into a multidimensional array
    // located at root node.

    function parse($xmlstring="")
    {
        // set up a new XML parser to do all the work for us
        $this->parser = xml_parser_create();
        xml_set_object($this->parser, $this);
        xml_parser_set_option($this->parser, XML_OPTION_CASE_FOLDING, false);
```

```
        xml_set_element_handler($this->parser, "startElement", "endElement");
        xml_set_character_data_handler($this->parser, "characterData");

        // Build a Root node and initialize the node_stack
        $this->node_stack = array();
        $this->startElement(null, "root", array());

        // parse the data and free the parser...
        xml_parse($this->parser, $xmlstring);
        xml_parser_free($this->parser);

        // recover the root node from the node stack
        $rnode = array_pop($this->node_stack);

        // return the root node
        return($rnode);
    }

    //---------------------------------------------------------------------

    // PROTECTED
    // Start a new Element. This is done by pushing the new element onto the stack
    // and reseting its properties.

    function startElement($parser, $name, $attrs)
    {
        // create a new node
        $node = array();
        $node["_NAME"] = $name;

        foreach ($attrs as $key => $value) {
                $node[$key] = $value;
        }

        $node["_DATA"] = "";
        $node["_ELEMENTS"] = array();
        // add the new node to the end of the node stack
        array_push($this->node_stack, $node);
    }

    //---------------------------------------------------------------------

    // PROTECTED
    // End an element. This is done by popping the last element from the
    // stack and adding it to the previous element on the stack.

    function endElement($parser, $name)
    {
        // pop this element off the node stack
        $node = array_pop($this->node_stack);
        $node["_DATA"] = trim($node["_DATA"]);
        // and add it an element of the last node in the stack...
        $lastnode = count($this->node_stack);
        array_push($this->node_stack[$lastnode-1]["_ELEMENTS"], $node);
    }

    //---------------------------------------------------------------------

    // PROTECTED
```

```
    //Collect the data onto the end of the current chars.

    function characterData($parser, $data)
    {
        // add this data to the last node in the stack...
        $lastnode = count($this->node_stack);
        $this->node_stack[$lastnode-1]["_DATA"] .= $data;
    }

    //-------------------------------------------------------------------
}

//#############################################################
//## END OF CLASS
//#############################################################

//sends Content-type header to client browser

function send_headers($encoding)
{
    Header("Content-type: text/html;charset=$encoding");
}

?>
```

8.4.Use Case in ASP: Searching SIETS Storage and Returning Results in HTML

This application is implemented in the ASP programming language.

The application searches the SIETS storage and returns the results in HTML.

```
<%

'
' This application is implemented in the VBScript programming language.
'
' The application searches the SIETS storage using HTTP API and returns the
' results in HTML.
'

%>
<html>
<head>
<title>Search</title>
<style>
#results div.header { margin-bottom: 35px; }
#results div.result { padding-left: 15px; }
#results p.title { margin-bottom: 3px; }
#results p.snip { margin: 0px; }
#results p.link { margin-top: 3px; font-size: 0.9em; color: gray; }
#results p.error { color: red; }
#results .pagelist { padding-top: 20px; }
#results .pagelist p { display: inline; }
#results .pagelist ul { margin: 0px; padding: 0px; display: inline; }
#results .pagelist li {    display: inline; }
</style>
```

```
</head>
<body>
    <div id="results">
<%

nDocs = 10 'results per page
nPages = 10 'pages listed

Offset = Int(Request.QueryString("page")) * nDocs 'pages are numbered from 0, displayed from
1
sQuery = Request.QueryString("query")

Set Http = Server.CreateObject("MSXML2.ServerXMLHTTP")
Http.Open "POST", "http://127.0.0.1/cgi-bin/siets/api.cgi", False
Http.Send "storage=test&command=search&docs=" & nDocs & "&offset=" & Offset &
"&relevance=yes&query=" & Server.URLEncode(sQuery)

if Http.Status = 200 and not Http.ResponseXML is Nothing then
    Set Dom = Http.ResponseXML
    Dom.SetProperty "SelectionNamespaces", "xmlns:s='www.siets.net'"
    Set Content = Dom.SelectSingleNode("s:reply/s:content")
    if not Content is Nothing then
        'command executed ok
        n = Int(Content.SelectSingleNode("hits").Text)
        %><div class="header"><p>Found <b><% if n > 0 then Response.Write n else
Response.Write "no"
        %></b> document<% if n <> 1 then Response.Write "s"
        if not Content.SelectSingleNode("real_query") is Nothing then sRealQuery =
Content.SelectSingleNode("real_query").Text else sRealQuery = ""
        %> matching &amp;quot;<%= Server.HTMLEncode(sRealQuery)
        %>&amp;quot; (<b><%= Dom.SelectSingleNode("s:reply/s:seconds").Text %></b>
seconds)</p></div><%= vbCrLf %><%
        if n > 0 then
            'something has been found
            for each Result in Content.SelectNodes("results/document")
                %><div class="result"><%
                sTitle = Result.SelectSingleNode("title").Text
                p = InStr(sTitle, ":_:_:")
                if p > 0 then sTitle = Left(sTitle, p - 1)
                %><p class="title"><a href="<%=
Server.HTMLEncode(Result.SelectSingleNode("id").Text) %>"><%= Server.HTMLEncode(sTitle)
%></a></p><%
                %><p class="snip"><%= Replace(Result.SelectSingleNode("text").Text,
"#", " ") %></p><%
                %><p class="link"><%=
Server.HTMLEncode(Result.SelectSingleNode("id").Text) %></p><%
                %></div><%= vbCrLf %><%
            next
            'page listing
            iFrom = Int(Content.SelectSingleNode("from").Text)
            nMore = Int(Mid(Content.SelectSingleNode("more").Text, 2))
            nSure = Int((nMore + iFrom + 2 * nDocs - 1) / nDocs)
            if iFrom > 0 or nMore > 0 then
                %><div class="pagelist"><%= vbCrLf %><p>Result pages:<%= vbCrLf
%><%
                iPage = Int(iFrom / nDocs)
                i = Int((iPage - 1) / (nPages - 2)) * (nPages - 2)
                if i < 0 then i = 0
                %><ul><%= vbCrLf %><%
```

```
                        sLink = "<a href=""search.asp?query=" & Server.URLEncode(sQuery) &
"&page="
                        if iPage > 0 then
                                %><li><%= sLink & (iPage - 1) %>">&lt;&lt;&lt;
Previous</a></li><%= vbCrLf %><%
                        end if
                        j = i
                        do while j < i + nPages and j < nSure
                                %><li><%
                                if j = iPage then
                                        %><b><%= j + 1 %></b><%
                                else
                                        %><%= sLink & j %>"><%= j + 1 %></a><%
                                end if
                                %></li><%= vbCrLf %><%
                                j = j + 1
                        loop
                        if nMore > 0 then
                                %><li><%= sLink & (iPage + 1) %>">Next
&gt;&gt;&gt;</a></li><%= vbCrLf %><%
                        end if
                        %></ul><%= vbCrLf %></p><%= vbCrLf %></div><%= vbCrLf
%><%
                end if
        end if
    else
        'error
        Set Content = Dom.SelectSingleNode("s:reply/s:error")
        %><p class="error">Error <%= Content.SelectSingleNode("code").Text %>: <%=
Server.HTMLEncode(Content.SelectSingleNode("text").Text) %></p><%= vbCrLf %><%
    end if
else
    %><p class="error">Search failed!</p><%= vbCrLf %><%
end if

%></div>
</body>
</html>
```

8.5.Use Case in Java: Searching SIETS Storage from applet

This application is implemented as Java applet.

The application searches the SIETS storage and displays results.

8.5.1.SietsJApi.java

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.Random;

public class SietsJApi extends JApplet implements ActionListener {
   private JPanel contentPane;
   private JPanel Buttons = new JPanel();
   private JPanel Results = new JPanel();
   private JPanel Properties = new JPanel();
```

```
   private JPanel MainPan = new JPanel();
   private JLabel HostLabel = new JLabel("Host:");
   private JLabel StorageLabel = new JLabel("Storage:");
   private JLabel QueryLabel = new JLabel("Query:");
   private JButton SearchButt = new JButton("Search");
   private JButton ClearButt = new JButton("Clear");
   private JTextField QueryField = new JTextField(10);
   private JTextField HostField = new JTextField("http://",20);
   private JTextField StorageField = new JTextField(10);
   private JTextArea ResultArea = new JTextArea();

   public void init() {
      //ContentPane Layout
      contentPane = (JPanel) this.getContentPane();
      contentPane.setLayout(new BorderLayout());

      //Main pane
      MainPan.setLayout(new GridBagLayout());
      //Properties pane
      Properties.setLayout(new GridBagLayout());
      Properties.setBorder(BorderFactory.createTitledBorder("Properties"));
      //Buttons pane
      Buttons.setLayout(new GridLayout(1,2,5,0));
      //Results pane
      GridLayout gridLayout1 = new GridLayout();
      gridLayout1.setVgap(0);
      gridLayout1.setHgap(0);
      gridLayout1.setColumns(1);
      gridLayout1.setRows(10);
      Results.setLayout(new BorderLayout());
      Results.setBorder(BorderFactory.createTitledBorder("Results"));
      //Add Main pane to contentPane
      contentPane.add(MainPan,BorderLayout.NORTH);

      //Properties pan to Main pane
      MainPan.add(Properties,new GridBagConstraints(0, 0, 1, 1, 1.0, 1.0
         ,GridBagConstraints.NORTH, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0,
0));
      //Add controls to Properties Pane
      Properties.add(HostLabel,new GridBagConstraints(0, 1, 1, 1, 1.0, 1.0
         ,GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(1, 1, 1, 1), 1, 0));
      Properties.add(HostField,new GridBagConstraints(1, 1, 1, 1, 1.0, 1.0
         ,GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0,
0));
      Properties.add(StorageLabel, new GridBagConstraints(2, 1, 1, 1, 1.0, 1.0
         ,GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(1, 1, 1, 1), 1, 0));
      Properties.add(StorageField, new GridBagConstraints(3, 1, 1, 1, 1.0, 1.0
         ,GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0,
0));
      Properties.add(QueryLabel, new GridBagConstraints(0, 2, 1, 1, 1.0, 1.0
         ,GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(1, 1, 1, 1), 0, 0));
      Properties.add(QueryField, new GridBagConstraints(1, 2, 3, 1, 1.0, 1.0
         ,GridBagConstraints.WEST, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0,
0));
      //Add Buttons to Main Pan
      MainPan.add(Buttons,new GridBagConstraints(0, 1, 1, 1, 1.0, 1.0
         ,GridBagConstraints.NORTH, GridBagConstraints.NONE, new Insets(1, 1, 1, 1), 20, 0));
      Buttons.add(SearchButt,null);
      Buttons.add(ClearButt,null);
```

```
      MainPan.add(Results,new GridBagConstraints(0, 2, 1, 1, 1.0, 1.0
         ,GridBagConstraints.NORTH, GridBagConstraints.BOTH, new Insets(1, 1, 0, 1), 0,0));

      Results.add(ResultArea);
      SearchButt.addActionListener(this);
      ClearButt.addActionListener(this);
   }

   //Action listener search and clear buttons
   public void actionPerformed(ActionEvent e) {
      if (e.getSource() == ClearButt) {
         QueryField.setText("");
      } else if (e.getSource() == SearchButt)  {
         String cgiUrl = new String(HostField.getText());
         SietsExch sietsReq;

         //Create Siets XML query
         SietsMess sietsXMLQuery = new
SietsMess("search",StorageField.getText(),"guest","guest",QueryField.getText());

         //Do data exchange with Siets server
         sietsReq = new SietsExch(cgiUrl,sietsXMLQuery.getMess());

         sietsReq.doQuery();

         String temp = sietsReq.getResponse();

         //Parse out XML results
         SietsXMLParser sietsXMLResp = new SietsXMLParser(temp.trim());

         String[][] resArray = new String[10][];

         resArray = sietsXMLResp.parse();
         String outp = "";

         //Format output
         for (int i = 0; i < sietsXMLResp.getResultLength(); i++) {
            System.out.println("URL["+i+"]: "+resArray[i][1]+" Title: "+resArray[i][0]);
            JLabel u;

            outp += "Title: "+resArray[i][0]+"\n";
            outp += "Link: "+resArray[i][1]+"\n\n";
         }

         ResultArea.setText(new String(outp));

      }
   }

}
```

8.5.2.SietsMess.java

```
import java.util.Calendar;

public class SietsMess {
   private String iComm;
   private String iData;
```

```
   private String iUser;
   private String iPass;
   private String iStorage;
   private String message;

   /** Creates new SietsMess */
   public SietsMess(String command,String storage,String user, String passwd) {
      iComm = command;
      iData = null;
      iUser = user;
      iPass = passwd;
      iStorage = storage;

      message = ComposeMess();

   }

   public SietsMess(String command,String storage, String user, String passwd,String data) {
      iComm = command;
      iData = data;
      iUser = user;
      iPass = passwd;
      iStorage = storage;
      message = ComposeMess();
   }

   public String getMess() {
      return message;
   }

   private String ComposeMess() {
      String mess = "";
      long current = System.currentTimeMillis();
      mess =  "<siets:request xmlns:siets=\"www.siets.net\">";
      mess +=
"<siets:timestamp>"+Calendar.YEAR+"/"+Calendar.MONTH+"/"+Calendar.DAY_OF_MONTH+"
"+Calendar.HOUR+":"+Calendar.MINUTE+":"+Calendar.SECOND+"</siets:timestamp>";
      mess += "<siets:command>"+iComm+"</siets:command>";
      mess += "<siets:requestid>"+current+"</siets:requestid>";
      mess += "<siets:storage>"+iStorage+"</siets:storage>";
      mess += "<siets:reply_charset>UTF-8</siets:reply_charset>";
      mess += "<siets:user>"+iUser+"</siets:user>";
      mess += "<siets:password>"+iPass+"</siets:password>";
      mess += "<siets:application>JavaApi</siets:application>";
      mess += "<siets:content>";
      if (iComm == "search") {
         mess += "<query>"+iData+"</query>";
         mess += "<docs>10</docs>";
      }
      mess += "</siets:content>";
      mess += "</siets:request>";
      return mess;
   }

}
```

8.5.3.SietsExch.java

```
import java.io.*;
```

```
import java.net.*;
import javax.swing.*;

public class SietsExch {
   private String iHost;
   private String iData;
   private String query;
   private String response;
   private String iFname;

   /** Creates new siets_network */
   public SietsExch(String Host, String data)  {
      iData = data;
      URL aURL=null;
      try {
         aURL = new URL(Host);
      } catch (MalformedURLException e) {
         System.out.println("Malformed URL");
      }
      iHost = aURL.getHost();
      iFname = aURL.getFile();

   }

   public int doQuery() {

      try {
         Socket socket = new Socket(iHost,80);
         BufferedWriter wr = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
         socket.setSoTimeout(60000); // set 1 minute timeout
         String header =  "POST "+iFname+"  HTTP/1.0\r\n";
               header += "Host: "+iHost+"\r\n";
               header += "User-Agent: SIETS Client Sample\r\n";
               header += "Content-Length: " + iData.getBytes("UTF-8").length+"\r\n\r\n";

   wr.write(header);
         wr.write(iData);

         wr.flush();

   response = read_socket(socket);

   wr.close();
         socket.close();

      } catch (UnknownHostException e) {
         System.err.println("Exception: Unknown host " + iHost + "!");
         System.exit(1);
      } catch (IOException e) {
         System.err.println("Exception: I/O error during connection!");
         System.exit(1);
      }

      return 1;

   }
```

```
   public String getResponse() {
      return response;
   }

   private String read_socket(Socket fsocket){
    String reply="";
    try{
         BufferedReader rd= new BufferedReader(new
InputStreamReader(fsocket.getInputStream()));

         StringBuffer tempresp = new StringBuffer();

         int ch=0;

         while (true){
            ch = rd.read();
            if (ch < 0)
               break;
            else
               tempresp.append((char)ch);
         }

         reply = new String(tempresp);

         reply = reply.substring(reply.indexOf("\r\n\r\n"));


    } catch (InterruptedIOException e){
         return
"<siets:reply>\n<siets:error>\n<text>"+e.getMessage()+"</text><source>API</source><leve
l>failed</level>\n</siets:error>\n</siets:reply>";
    } catch (UnknownHostException e){
         return
"<siets:reply>\n<siets:error>\n<text>"+e.getMessage()+"</text><source>API</source><leve
l>failed</level>\n</siets:error>\n</siets:reply>";
    } catch (IOException e){
         return
"<siets:reply>\n<siets:error>\n<text>"+e.getMessage()+"</text><source>API</source><leve
l>failed</level>\n</siets:error>\n</siets:reply>";
    } catch (NullPointerException e){
         return
"<siets:reply>\n<siets:error>\n<text>"+e.getMessage()+"</text><source>API</source><leve
l>failed</level>\n</siets:error>\n</siets:reply>";
    }
   return reply;
}

}
```

8.5.4.SietsXMLParser.java

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;

public class SietsXMLParser {
   private String iData;
   private String[][] Resultset = new String[10][2];
```

```
   private int ResCount = 0;

   /** Creates new SietsXMLParser */
   public SietsXMLParser(String data) {
      iData = data;
   }

   public String[][] parse() {
      try {
         DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
         Document doc = builder.parse(new InputSource(new StringReader(iData)));

         NodeList nodes = doc.getElementsByTagName("document");

         ResCount = nodes.getLength();

         for (int i = 0; i < nodes.getLength(); i++) {
            Element element = (Element) nodes.item(i);

            NodeList title = element.getElementsByTagName("title");
            Element line = (Element) title.item(0);

            Resultset[i][0] = new String(getCharacterDataFromElement(line));

            NodeList url = element.getElementsByTagName("id");
            line = (Element) url.item(0);

            Resultset[i][1] = new String(getCharacterDataFromElement(line));
         }

      } catch (Exception e) {
         e.printStackTrace();
         return null;
      }
      return Resultset;
   }
   public int getResultLength() {
      return ResCount;
   }
   public String getCharacterDataFromElement(Element e) {
      Node child = e.getFirstChild();
      if (child instanceof CharacterData) {
         CharacterData cd = (CharacterData) child;
         return cd.getData();
      }
      return "?";
   }
}
```

8.6.Use Case in FoxPro I: Updating SIETS Storage from Database Table Data

This application is implemented in the Microsoft Visual FoxPro programming language.

The application reads data from a database table and imports them to the SIETS storage.

Note: In this use case, a simple function called XTAGC for searching XML tags is used instead of the fetch functions.

```
PROCEDURE siets_update

*!*      variables for the SIETS session initiation parameters
LOCAL lcAplication,lcServer,lnPort,lcStorage,lcUsr,lcPasswd;

lcAplication=[siets_update]     && application name
lcServer=[111.111.111.111]      && IP adress of the server
lnPort=1111                     && port number
lcStorage=[mani-dati]           && storage name
lcUsr=[my-user]                 && user name
lcPasswd=[my-passwd]            && user password
lcTimeout=60                    && timeout

LOCAL lcReplyXml,lcError,lcSession

*!* initialization of SIETS API library
SIETS_DECLARE()

*!* initialization of SIETS parameters
lcSession = SIETS_OPEN_SESSION(lcServer,lnPort,lcStorage,lcUser,lcPasswd,lcTimeout)

*!* if the error ocured, returns
IF lcSession < 0
    ? [Error opening session]
    RETURN .F.
ENDIF

*!* variables to which the indexing data will set
LOCAL lcText, lcTitle, lcLink, lnRateing, lcCharset

*!* encoding of data to be updated
lcCharset=[windows-1257]

*!* opens a table with data that are to be stored to the SIETS storage
USE dati ALIAS dati
SELECT dati
SET ORDER TO id
GO TOP

SCAN
    lcLink=ALLTRIM(dati.id)     && id
    lcTitle=ALLTRIM(dati.title) && title
    lcText=ALLTRIM(dati.text)   && textual information

    *!* rate is calculated by the updating date: the newer the document, the greater the rate
    lnRateing=dati.date-DATE(1970,1,1)

    *!* replaces the & and " characters with the XML entities
    lcTitle=STRTRAN(lcTitle,[&],[&amp;]);
    lcTitle=STRTRAN(lcTitle,["],[&quot;]);
    lcText=STRTRAN(lcText,[&],[&amp;]);
    lcText=STRTRAN(lcText,["],[&quot;]);

    *!* sends data to the SIETS storage
```

```
    lcReplyXml=SIETSXML_UPDATE(lcSession,lcLink,lcTitle,lnRateing,lcText,lcCharset)

    *!* parses the error message from the XML reply
    lcError=XTAGC([siets_error],lcReplyXml,1);

    IF NOT EMPTY(lcError)
        IF XTAGC([level],lcError,1)==[failed]
                        *!*     if the error level is failed, the system work can be continued,
                        *!*     because the error refers only to the input data
                ? [The update was not executed successfully for a document with id: ]+lcLink
                ? [Error message: ]+XTAGC([text],lcError,1)
        ELSE
                        *!*     if the error level is error or fatal, the function returns
                ? [The update is interrupted for a document with id: ]+lcLink
                ? [Error message: ]+XTAGC([text],lcError,1)
                RETURN .F.
        ENDIF
    ENDIF
ENDSCAN

*!* when the update is complete, the index command is executed
lcReplyXml=SIETS_INDEX(lcSession)

*!* parses the error message from the XML reply
lcError=XTAGC([siets_error],lcReplyXml,1);

*!* if the error message is not empty, returns
IF NOT EMPTY(lcError)
    ? [Error:] + lcError
    RETURN .F.
ENDIF

SIETS_CLOSE_SESSION(lcSession)

RETURN .T.

*!* additional function XTAGC, which is used in this use case for a easier XML parsing
*!* the XTAGC function returns a substring from a given string, which contains data
*!* between the tags <m.tag>...</m.tag>

PARAMETER m.t, m.s, m.o
PRIVATE m.i,m.j

STORE .F. TO m.XTAGC_ISTAG, m.ISTAG

IF EMPTY(m.s)
    RETURN ""
ENDIF
IF EMPTY(m.o)
    IF !EMPTY(m.t)
        m.i=ATC("<"+m.t+">",m.s)
        m.j=ATC("</"+m.t+">",m.s)
    ELSE
        m.i=ATC(">",m.s)
        m.j=ATC("</",m.s)
    ENDIF
ELSE
    IF !EMPTY(m.t)
        m.i=ATC("<"+m.t+">",m.s,m.o)
```

```
        m.j=ATC("</"+m.t+">",m.s,m.o)
    ELSE
        m.i=ATC(">",m.s)
        m.j=ATC("</",m.s)
    ENDIF
ENDIF

IF m.i=0 OR m.j=0
    RETURN ""
ENDIF

STORE .T. TO m.XTAGC_ISTAG, m.ISTAG

m.i=m.i+LEN(m.t)+IIF(!EMPTY(m.t),2,1)

IF SUBSTR(m.s,m.i,2)==CHR(13)+CHR(10)
    m.i=m.i+2
ENDIF

IF SUBSTR(m.s,m.j-2,2)==CHR(13)+CHR(10)
    m.j=m.j-2
ENDIF
IF m.j>m.i
    RETURN SUBSTR(m.s,m.i,m.j-m.i)
ENDIF
RETURN ""
```

8.7.Use Case in FoxPro II: Searching SIETS Storage and Returning Results to Cursor

This application is implemented in the Microsoft Visual FoxPro programming language.

The application searches the SIETS storage and returns the results to the cursor.

Note: In this use case, a simple function called XTAGC for searching XML tags is used instead of the fetch functions.

```
PROCEDURE siets_search
*!* parameters:
*!* tcCursorName – name of the cursor to which the search results will be returned
*!* tnMaxDoc – number of results to be returned
*!* tnOffset – intend from the beginning of the result set
*!* tcOrder – mechanism by which results are to be sorted: by relevance (.T.), or rate (.F.)
*!* tcErrorMsg – returns error message
*!* tcQuery – search query

LPARAMETERS tcCursorName, tnMaxDoc, tnOffset, tlOrder, tcErrorMsg, tcQuery
*!* variables for the SIETS session initiation parameters
LOCAL lcAplication,lcServer,lnPort,lcStorage,lcUsr,lcPasswd;

lcAplication=[siets_update]     && application name
lcServer=[111.111.111.111]      && IP address of the server
lnPort=1111                     && port number
lcStorage=[mani-dati]           && storage name
lcUsr=[my-user]                 && user name
lcPasswd=[my-passwd]            && user password
lcTimeout=60                    && timeout
```

```
LOCAL lcReplyXml,lcError,lcSession

*!* initialization of SIETS API library
SIETS_DECLARE()

*!* initialization of SIETS parameters
lcSession = SIETS_OPEN_SESSION(lcServer,lnPort,lcStorage,lcUser,lcPasswd,lcTimeout)

*!* if the error ocured, returns
IF lcSession < 0
    ? [Error opening session]
    RETURN .F.
ENDIF

*!* creation of the cursor where to return the search results
CREATE CURSOR (tcCursorName) (link C(10) , Title C(254), date D, Highlight M)

LOCAL lcReplyXml,lcError,lnFound,lcCaseSensitive,lcRelevance,lcFromDomain
lcRelevance=.F.
lcFromDomain=0
lcCaseSensitive=.F.

*!* performs FTS with the given search parameters
lcReplyXml=SIETSXML_SEARCH(lcSession,tcQuery,tnMaxTrans,tnOffset,lcRelevance,lcFromD
omain,lcCaseSensitive,[windows-1257]) && Perform search

*!* parses an error message from the XML reply
lcError=XTAGC([siets_error],lcReplyXml,1)

*!* if the error message is not empty, returns
IF NOT EMPTY(lcError)
    tcErrorMsg=XTAGC([text],lcError,1) && returns the error message
    RETURN .F.
ENDIF

*!* parses the number of results from the XML reply
lnFound=VAL(XTAGC("found",lcReplyXml,1))

LOCAL lnI,lclink,lnRate,lcTitle,lcText,ldDate
*!* parses the search results from the XML reply and saves them in the cursor
FOR lnI=1 TO lnFound STEP 1
    lcLink=XTAGC([link],lcReplyXml,lnI)                 && id
    lnRate=VAL(XTAGC([rate],lcReplyXml,lnI))            && rate
    lcTitle=XTAGC([title],lcReplyXml,lnI)               && title
    lcText=XTAGC([text],lcReplyXml,lnI)                 && snippet of text with search terms
    ldDate=DATE(1970,1,1)+lnRate            && rate converted back to the date format
    *!* saving data to the cursot
    INSERT INTO (tcCursorName) (link,Title,date,Highlight) VALUES
(lcLink,lcTitle,ldDate,lcText)
ENDFOR

SIETS_CLOSE_SESSION(lcSession)

RETURN .T.

*!* additional function XTAGC, which is used in this use case for a easier XML parsing
*!* the XTAGC function returns a substring from a given string, which contains data
*!* between the tags <m.tag>...</m.tag>
```

```
PARAMETER m.t, m.s, m.o
PRIVATE m.i,m.j

STORE .F. TO m.XTAGC_ISTAG, m.ISTAG

IF EMPTY(m.s)
    RETURN ""
ENDIF
IF EMPTY(m.o)
    IF !EMPTY(m.t)
        m.i=ATC("<"+m.t+">",m.s)
        m.j=ATC("</"+m.t+">",m.s)
    ELSE
        m.i=ATC(">",m.s)
        m.j=ATC("</",m.s)
    ENDIF
ELSE
    IF !EMPTY(m.t)
        m.i=ATC("<"+m.t+">",m.s,m.o)
        m.j=ATC("</"+m.t+">",m.s,m.o)
    ELSE
        m.i=ATC(">",m.s)
        m.j=ATC("</",m.s)
    ENDIF
ENDIF

IF m.i=0 OR m.j=0
    RETURN ""
ENDIF

STORE .T. TO m.XTAGC_ISTAG, m.ISTAG

m.i=m.i+LEN(m.t)+IIF(!EMPTY(m.t),2,1)

IF SUBSTR(m.s,m.i,2)==CHR(13)+CHR(10)
    m.i=m.i+2
ENDIF

IF SUBSTR(m.s,m.j-2,2)==CHR(13)+CHR(10)
    m.j=m.j-2
ENDIF
IF m.j>m.i
    RETURN SUBSTR(m.s,m.i,m.j-m.i)
ENDIF
RETURN ""
```

APPENDIX A: ERROR MESSAGES

This section contains a list of error messages and their codes.

Code	Error message content	Description
1001	XML Parsing error.	Error that occurred when parsing the XML request. Check the XML syntax for your input data.
1002	Missing required tag.	Error the occurred due to missing of required information in the input data, for example, a document id, or the result count docs in the search request.
1003	Wrong SIETS storage.	Error occurred due to incorrect name of the SIETS storage. Check if the SIETS storage name, IP address, and the port are correct.
1004	Invalid user name.	Error occurred due to incorrectly entered user name. For information on user administration, see the *SIETS Administration and Configuration Guide*.
1005	User not allowed to login from that IP.	Error occurred due to restriction to login from that IP. For information on user administration, see the *SIETS Administration and Configuration Guide*.
1006	Incorrect password.	Error occurred due to incorrectly entered password. For information on user administration, see the *SIETS Administration and Configuration Guide*.
1008	Invalid command name.	Error occurred due to incorrectly entered command name. Check the command name and if the SIETS API version corresponds to the SIETS server version.
1009	System is busy.	Error occurred due to a massive load. Check that there are enough resources for the SIETS server, or that there are no hardware problems.
1010	Subsystem not responding.	Error occurred due to a fatal error. Check if all SIETS server processes are running. For more information on SIETS server processes, see the *SIETS Administration and Configuration Guide.*
1012	Requested document does not exist.	Error occurred due to that a document with the ID does not exist in the SIETS storage for functions like replace, delete, or retrieve.
1013	Memory allocation failed.	Error occurred due to an internal error when trying to allocate memory. Check if there is enough RAM for the SIETS server.

Code	Error message content	Description
1014	Disk I/O error.	Error occurred due to an input or output error. Check your hard disk for a damage or occupied disk space overflow.
1016	Internal error.	Error occurred due to a memory corruption. This is a fatal error, possibly caused by a hardware damage, kernel panic, or software error.
1021	Document with that ID already exists.	Error occurred due to that a document with such ID already exists in the SIETS storage for the function insert.
1022	Corrupted data.	Error occurred due to an internal error in the SIETS storage. It can be caused by hardware damage or a severe software error.

For different SIETS versions the list of errors may slightly differ.

Reporting Problems

Problems that occur when working with SIETS can be reported to SIETS technical support.

To report a problem, provide the following to the technical support:

- name of your company
- SIETS license number
- version of the SIETS server
- SIETS log files

For contact information on technical support, see Contact Information.

For information on the SIETS log files, see the *SIETS Administration and Configuration Guide*.

APPENDIX B: API REFERENCE FOR FOXPRO

This section contains API reference for Microsoft Visual FoxPro programming language.

Almost all modern programming languages have built-in functions or libraries with functions with implement HTTP protocol. However, Microsoft Visual FoxPro does not have such built-in functions or libraries with such functions. Therefore, functions in Microsoft Visual FoxPro have been implemented, using which you can call SIETS API commands.

This section contains the following topic:

- Microsoft Visual FoxPro

Microsoft Visual FoxPro

Use the siets_declare() function to declare the SIETS dynamic linkage library siets_api.dll that implements the SIETS API library functions.

Library Initiation

```
siets_declare()
```

Session

```
siets_open_session(cHost, nPort, cStorage, cUser, cPasswd, nTimeout)
siets_close_session(nSessId)
```

Generic Command

```
siets_command(nSessId, cCmdName, cContent, cEncoding)
```

Data Manipulation

```
siets_insert(nSessId, cId, cTitle, nRate, cDomain, cText, cInfo, cHidden, cEncoding)
siets_update(nSessId, cId, cTitle, nRate, cDomain, cText, cInfo, cHidden, cEncoding)
siets_replace(nSessId, cId, cTitle, nRate, cDomain, cText, cInfo, cHidden, cEncoding)
siets_delete(nSessId, cId, cEncoding)
siets_index(nSessId)
siets_clear(nSessId)
siets_get_sheme(nSessId)
siets_set_sheme(nSessId, aLocations, aPolicies)
```

Status Monitoring

```
siets_status(nSessId)
```

Data Retrieval

```
siets_lookup(nSessId, cId, cEncoding)
siets_retrieve(nSessId, cId, cEncoding)
siets_search(nSessId, cQuery, nDocs, nOffset, cRelevance, nMaxFromDomain, cCaseSupport,
nRateFrom, nRateTo, cEncoding)
siets_similar(nSessId, cId, cText, nLen, nQuota, nDocs, nOffset, cEncoding)
siets_alternatives(nSessId, cQuery, cEncoding)
```

APPENDIX C: FREQUENTLY ASKED QUESTIONS

This section contains the following frequently asked questions:

- How can I import binary data to SIETS?
- How can I make SIETS to automatically ignore common words when performing FTS?
- How can I see the actual query that used for FTS?
- How can I export the vocabulary with an SIETS API command?
- Why do I get an error: connection failed when importing data to the SIETS server from a Windows NT environment?
- Is it possible to return more than 1000 documents to the result set?
- When I import large amount of data to the SIETS storage, why are they not available for FTS for a while?

How can I import binary data to SIETS?

To import binary data like MS Word or PDF document files to the SIETS storage, they must be entered in the info document part.

Note: Data in the info part are not available for FTS. If you want your data to be available for FTS, they must be stored as plain text.

Usually, binary data do not comply with the XML formatting standard. However, to be imported to the SIETS storage, they must comply with the XML formatting standard. Therefore, before importing to the SIETS storage, you must encode the binary data to the base64 encoding or other.

For more information on document parts, see Understanding SIETS Document Structure.

How can I make SIETS to automatically ignore common words when performing FTS?

The SIETS server automatically detects words that appear in the SIETS storage most often and adds them to the ignored words list. These words are considered to be common words that are ignored during FTS.

It is possible to edit the limit of the ignored words list. For more information on managing the ignored word list limit, see the *SIETS Administration and Configuration Guide*.

For more information on ignored words, see Ignored Words.

How can I see the actual query that used for FTS?

Often the actual query that is used for FTS differ from that you entered as a search query. Reasons for this can be the following:

- If the original query contains words from the ignored words list, they are dropped from the actual search query.
- If the original query contains wildcard patterns, a class of words created from the wildcard pattern usage is entered in the actual search query.

To see the actual query used for FTS, use the <real_query> tag of the XML reply to the search command.

For more information on the search command, see Search.

How can I export the vocabulary with an SIETS API command?

The vocabulary is a list of all unique words in the SIETS storage. Unique words are found in documents and added to the vocabulary while storing these documents to the SIETS storage. Each SIETS storage has its own vocabulary.

Unfortunately, it is not possible to export the vocabulary with any of the SIETS API commands. However, on the file level the vocabulary is stored in a text file, where each line contains one word. You can copy this text file and view it.

For information on the vocabulary text file, see the *SIETS Administration and Configuration Guide*.

For more information on vocabulary, see Understanding Storing Information in SIETS.

Why do I get an error: connection failed when importing data to the SIETS server from a Windows NT 4.0 or Windows 2000 environment?

Importing data to the SIETS server, just like any other operation with the SIETS server, is performed by transporting XML requests and replies via HTTP.

When importing large amount of data to the SIETS server, many TCP/IP connections are opened. After the connections are closed, they remain in the TIME_WAIT state for a definite time period.

By default, in the Windows NT 4.0 or Windows 2000 environment, the limit of the connections is inconsiderably small and the TIME_WAIT state time period is too long.

Therefore, because the number of new connections created per second can be very large and the closed connections remain in the TIME_WAIT state for some time period, the number of connections can reach the limit very fast.

In that case, the system does not allow to create a new connection and the error is returned.

To configure the limit of the connections and the TIME_WAIT state time period, configure the following key in the Windows NT 4.0 or Windows 2000 registry:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters]

"TcpTimedWaitDelay"=dword:00000015
```

where, the value is a decimal number representing seconds.

Is it possible to return more than 1000 documents to the result set?

By default, the limit of documents to be returned to the result set is 1000. It is possible to increase this limit. However, there is the following functionality, which is designed for the maximum number of documents in the result set equal to 1000:

- sorting search results by the relevance
- grouping search results by a domain

If you increase the limit of documents in the result set, the limit will be applied for all functions, except, if sorting search results by relevance or domain, only 1000 documents will be returned to the result set.

If you increase the limit of documents in the result set, it means that transactions in the SIETS server will be performed in a longer time period. Therefore, you should also increase the timeout period of functions.

For more information on configuring the limit of documents in the result set, see the *SIETS Administration and Configuration Guide*.

For more information on the relevance, see Relevance.

For more information on grouping documents by a domain, see Search.

When I import large amount of data to the SIETS storage, why are they not available for FTS for a while?

When importing data to the SIETS storage, if the memory reserved for memory cache is not enough for the data amount being imported, then:

1. The data being imported are written to another cache, which is written to the disk, and the index state is `expanding`.
2. When the importing is complete, the SIETS server is committing data written on the disk to the inverted index, and the index state is `collapsing`.

While the index state is expanding or collapsing, the data written to the disk are not available for FTS. Only when data are added to the inverted index, they are available for FTS.

For example, if the data amount to be imported is tens of GB, these data will not be available for FTS for few hours.

For more information on the index state, see Status.

GLOSSARY

A	
API	Application programming interface.
B	
Boolean expression	Expression containing logical operators like AND, OR, and NOT.
C	
case-support	Feature that allows distinguishing between uppercase and lowercase characters.
D	
demon	Program or process, part of a larger program or process, that is dormant until a certain condition occurs and then is initiated to do its processing
document repository	Place where all SIETS documents are stored in the format, in which they were stored in the SIETS system, for returning the documents on a search request. See also: SIETS document.
F	
FTS	Full text search. See also: FTS query.
FTS query	Full text search request to the SIETS server. See also: FTS.
fuzzy search	Feature that allows searching for words that sound the same but are spelled differently.
I	
inverted index	List of words, where each word has a list of pointers to SIETS documents in which the word occurs. See also: SIETS document.
M	
markup search	Feature that allows searching for words within specific markup.
multi-language support	Feature that ensures the documents in different languages and character encodings can be stored and searched within one SIETS storage. See also: SIETS storage.
P	
policy	Set of operations for data importing and retrieving to the SIETS storage. See also: SIETS storage.
R	
RAM	Random access memory.
rate	Mechanism, which ensures that results are ordered in a result set according to assigned unchanging rate.
relevance	Measure of the accuracy of the search results. See also: rate.
S	
scheme	Document structure definition mechanism, which defines the location and behavior of each document part.

SIETS API	Standardized set of functions for accessing the SIETS server. See also: SIETS server.
SIETS console	Simple text application for accessing the SIETS server directly using the same functions as in SIETS API. See also: SIETS API.
SIETS document	Unit in SIETS storage against which searching is performed. It can be unstructured or XML structured. See also: SIETS storage.
SIETS server	Stand-alone server for storing and retrieving information such as plain texts or XML structured documents. It can be run in one or more instances per computer.
SIETS storage	Data collection for storing SIETS documents in a format that ensures a search is performed very fast. SIETS storage is contained by one SIETS server instance, and consists of vocabulary, document repository, and inverted index. See also: SIETS document, vocabulary, document repository, inverted index.
snippet	Fragment with an occurrence of a search term.
specific weight	Weight of a word in a document that is calculated according to the specific weight interval of the document part the word occurs. See also: specific weight interval.
specific weight interval	Integer range assigned to a document part that denotes the importance of the document part compared to other document parts. See also: specific weight.
stemming	Feature that allows searching for words and their declinations.
U	
UTF	UCS (universal character set) transformation format.
V	
vocabulary	Vocabulary is a list of all unique words in the SIETS storage. Unique words are found in documents and added to the vocabulary while storing these documents to the SIETS storage. See also: SIETS storage.
W	
wildcard search	Feature that allows searching for unknown characters or phrases.
X	
XML	Extensible markup language.
XML reply	XML message that is returned when submitting a XML request. See also: XML request.
XML request	XML message that is sent to the SIETS server to perform a SIETS API command. See also: SIETS API.

INDEX

A
abbreviations..... 7
API..... 102
audience..... 6, 97
C
case-support..... 102, 103
client — server architecture..... 14
concepts..... 12
creating.....
document structure with application..... 21
D
demon..... 102
document.....
repository..... 13, 102
document ordering..... 24
document part..... 26, 28
E
encoding.....
different..... 33
multiple bytes per character..... 32, 39
one byte per character..... 32, 34
UTF-8..... 30
encoding parameter..... 31
F
features..... 20
FTS..... 102
FTS query..... 102
I
importing.....
XML structured data..... 22
Internationalization..... 30
Introducing SIETS..... 8
inverted index..... 13, 102
M
multi-language support..... 30, 102
multi-server architecture..... 15
multiple storages architecture..... 14
P
policy..... 102
Preface..... 6
Q
querying.....
SIETS storage..... 18
R
RAM..... 102
rating..... 26
related information..... 7
relevance..... 26, 102
relevance.....
2.4.3.1.calculation algorithm..... 27
example..... 27
S
search.....
query..... 102
SIETS.....
and the future..... 20
API..... 12, 103
API specification..... 36
architecture..... 14
console..... 13, 103
document..... 13, 103
FTS capability..... 12
in corporate networks..... 9
server..... 12, 103
storage..... 13, 103
understanding.....
environment..... 10
5.SIETS clustering..... 64
creating..... 64
principles..... 64
SIETS console..... 65
single.....
encoding..... 32
snippet..... 103
specific weight..... 27
1.specific weight interval..... 26
customizing..... 29
SQL..... 103
standards compatibility..... 19
structure.....
of this guide..... 6
T
typographic conventions..... 7
U
understanding.....
SIETS document structure..... 21
storing information in SIETS..... 16
8.use cases..... 70
C
importing text files..... 70, 74
FoxPro.....
searching SIETS storage and returning results to cursor..... 93
updating SIETS storage from database table data..... 90

uses cases.....
PHP.....
searching SIETS storage and returning results in HTML.....77, 82, 84
UTF.....103
V
vocabulary..... 13, 103
W
What is SIETS?..... 8
X
XML..... 103
command.....38
element.....38
entity.....34
formatting..... 35
markup..... 35
message.....
structure.....38
reply.....103
request..... 103
special characters..... 35
XML command.....
alternatives..... 60
clear.....42
delete.....42
error handling..... 63
get_scheme.....43
index.....42
insert.....40
list-last..... 61
lookup.....46
search.....47
select..... 58
set_scheme.....43
similar..... 59
status.....44