Project ZNIX
Software Requirements Specification

Version 1.0

Revision History

Date	Version	Description	Author
2005/02/10	1.0	Initial Draft.	Sameera Perera

Table of Contents

Index of Tables and Figures

Software Requirements Specification

1. Introduction

This section introduces the System Requirements Specification (SRS) for Project ZNIX to its readers.

1.1 Purpose

The purpose of this software requirements specification (SRS) is to establish the major requirements of the Project ZNIX research and development effort.

Project ZNIX is an attempt to refine traditional information management systems to provide an object-based, application-independent retrieval and manipulation framework.

1.2 Scope

The SRS applies to the higher level design of the ZNIX Framework.

1.3 Definitions, Acronyms and Abbreviations

This section provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. Note that capitalization styles are used throughout the document to distinguish ZNIX-specific terms.

1.3.1 ZNIX

ZNIX is not an acronym. The term was decided upon to suggest the aspirant notion that a fully ZNIX integrated operating system could be the last (thus the letter “Z”) major evolution for POSIX system.

1.3.2 ZNIX Framework

The ZNIX Framework is a collection of system services (daemons) that:

- Facilitate user interactions with information and,
- Expose lower level functionality through a public API

Often, the terms Project ZNIX and the ZNIX Framework and Framework Shell would be used interchangeably at this stage.

1.3.3 Client Process

A client process is a software application that interacts with the ZNIX Framework by either utilizing the public API or by directly accessing the information presented by the Framework. Note that all client application **need not** be "ZNIX-aware" (i.e. knows that the information it accesses are managed by the Framework). As long as its operation involves an intervention by the Framework at some point, an application may be considered a ZNIX Client.

1.3.4 ZNIX Proxy Mechanism

The Proxy Mechanism is the collection of Framework components that enable "ZNIX-unaware" applications to utilize the Framework capabilities.

1.3.5 Entity

An Entity is the basic block that builds up ZNIX's information hierarchy. Entities are first class citizens of the ZNIX Framework; which simply means that the Identity of an Entity is unique and non-dependant on any other Entity.

1.3.6 Associate

An Associate is similar to an Entity, differing in the fact that its Identity is dependant on at least one Entity. The Framework's interest in an Associate depends solely on its relationship(s) to first class citizens. As such, when the last related Entity breaks the relationship to an Associated (i.e. loses interest in it), it will be removed from the ZNIX data[1] store as it no longer has any significance in the context.

[1] Note the use of the word "data" when it has been stated that the word "information" is preferred in this context. The thinking behind this is that, what the framework manages is raw data; it will associate a meaning to this data effectively transforming them into information.

1.4 References

- *[Zachary, 1994]:* Inspired by “Information at your finger tips”- Bill Gates, the vision behind Microsoft’s (abandoned) project Cairo. Quoted from Zachary, G. Pascal 1994. *Show-Stopper! The Breakneck Race to Create Windows NT and the Next Generation at Microsoft*. Little, Brown and Company (UK). (Note: Longhorn, the upcoming successor to Windows XP is said to be a continuation of Project Cairo).
- *WWW01*: Project Haystack website. http://haystack.lcs.mit.edu/
- *WWW02*: ZDNet Software Review website. http://reviews-zdnet.com.com/4514-3513_16-21008729.html?tag=subnav
- WWW03: Open Source Initiative Website.

  http://opensource.org/licenses/lgpl-license.php
- WWW04: Open Source Initiative Website.

  http://opensource.org/licenses/gpl-license.php
- *WWW05*: MSDN Website http://msdn.microsoft.com/winlogo/
- *WWW06:* Websites for Unicode Standards

  www.unicode.org

  ftp.informatik.uni-erlangen.de/pub/doc/ISO/charsets/ISO-10646-UTF-8.html

  ftp.informatik.uni-erlangen.de/pub/doc/ISO/charsets/ISO-10646-UTF-16.html
- Special Notes

  The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

The words “He” or “he”, whenever not referring to a nominal individual, MAY be read as “he or she” throughout this documentation.

1.5 Overview

This specification is organized into the following sections:

- *Introduction*, which introduces the specification for Project ZNIX to its readers.
- *Overall Description*, which provides a brief, high level description of Project ZNIX including its definition, benefits over alternatives, capabilities etc.
- *Specific Requirements*, which describes the software requirements to a level of detail sufficient to facilitate design and testing processes.

2. Overall Description

Project ZNIX envisions a world which leaves behind a one that users manage information by mastering software applications [Zachary, 1994]. Emphasis in this approach is on information[2] and its semantics. It seeks to make meaningful information accessible to users in ways they themselves declare to be logical.

2.1 Project Perspective

Project ZNIX SHOULD distinguish itself form other information management systems by its superior integration with the operating system. The Framework, in its operation, SHOULD resemble the schematic diagram displayed in Figure 1.3.6-A.

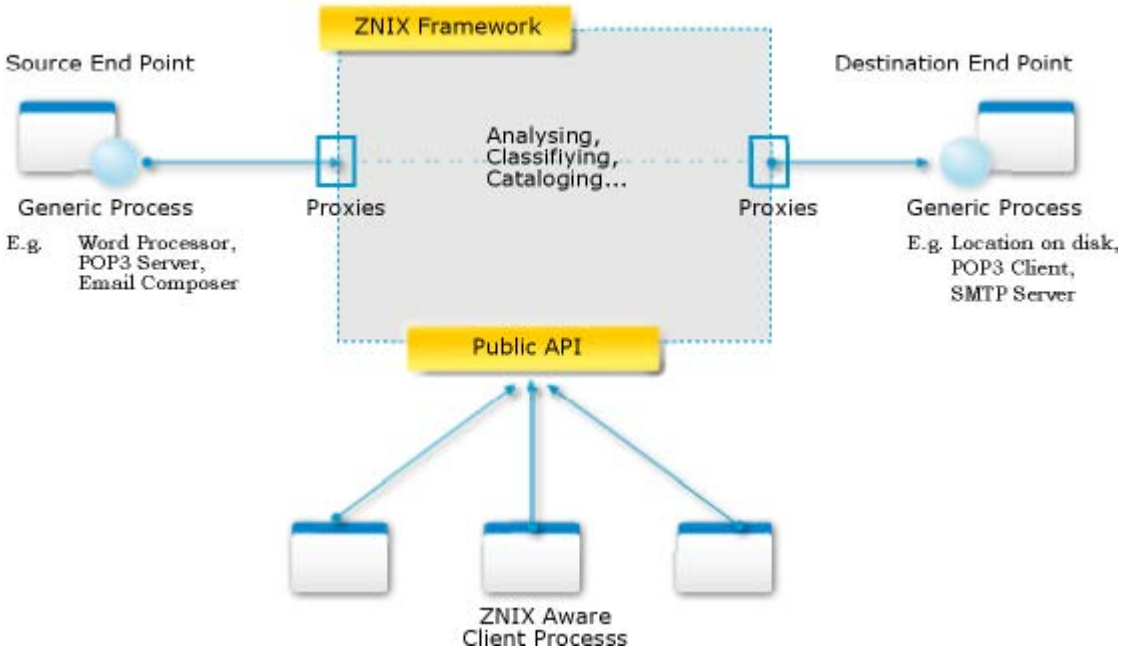

Figure 1.3.6-A

[2] In ZNIX terminology the word “Information” is always preferred over the word “data”, as the former implies that there is an associated “meaning”.

2.1.1 Benefits over alternatives

2.1.1.1 Project Haystack

"Haystack is a tool designed to let every individual manage all of their information in the way that makes the most sense to them" [WWW01].

Sharing the idea of a centralized store for all types of information with ZNIX, Project Haystack provides a tool (an extendable browser) for accessing the managed information. This tool is however, a standalone application with little support for 3rd party applications. Although, it does provide means for developers build additional, unless the project is adopted as a global standard there would be little incentive for the development of (especially commercial) extensions.

ZNIX includes both an API and a proxy mechanism. The API would enable developers to build applications that utilize the full potential of the Framework. The proxy mechanism would enable many established 3rd party applications to interact with the Framework.

Requirements to run Haystack on an Intel PC include 512 megabytes of RAM (with 768 megabytes strongly being recommended). ZNIX Framework SHOULD run comfortably on systems with only 256 megabytes of RAM.

2.1.1.2 Windows Longhorn

Project ZNIX was largely inspired by an ancestor of Microsoft's Windows Longhorn. However, since at the time of this specification, features inherent in this operating system have not been finalized, little conscious effort will be taken to incorporate them into ZNIX.

It has been reported that "by the time Longhorn ships, according to Microsoft chairman Bill Gates, PCs will have 4GHz to 6GHz processors, more than 2GB of memory, at least a terabyte of storage, and graphics accelerators three times more powerful than those offered by ATI and NVIDIA today" [WWW02]. ZNIX on the other hand is designed from the ground up to run on system with far less resources.

2.2 Product Functions

- Provide a centralized archive for user's information.
- Provide hierarchical model for managing the archived information. Hierarchies maybe inheritance based, association based or based on any other type of relationship conceived by the user.
- Allow the user to add, remove, modify or back up (to removable storage devices) arbitrary pieces of information without disrupting the hierarchy.
- Assist the user in building up his information hierarchy by automating such tasks as data capturing and categorizing.
- Provide a simple API for developing ZNIX-aware client applications.

2.3 User Characteristics

The browser GUI SHOULD be designed such that even casual users could easily manipulate their information with little or no training. It is expected that the bulk of the ZNIX user-base would be home users, managerial staff, executives, and researchers who are likely to possess only basic IT skills.

2.4 Constraints

The system design MUST be subjected to the following constraints:

- Hardware: Intel Pentium III or compatible 1-GHz PC with 256 megabytes of RAM

2.5 Assumptions and Dependencies

The system design MAY assume that all interfaces designed for the 3rd party software listed in Table 3.9.3-A will be compatible with their subsequent releases. Thus, for instance, when the system is found to be compatible to Windows 2000 Professional Service Pack 4, it MAY be assumed to be Windows 2000 Professional Compliant, regardless whether other service releases would break the system.

The applications listed in this table are assumed to be safe from vulnerabilities. The security offered by these products will form the basis of the security of the Framework.

It is assumed that supporting the software listed would be sufficient to serve the intended user-base, considering their popularity on Microsoft Windows based systems.

3. Specific Requirements

3.1 Functionality

This section describes the functional requirements of the system.

3.1.1 Centralized store for user information

The Framework MUST be capable of capturing information form the software tools specified in Table 3.9.3-A. The data captured SHOULD be stored in a centralized data store. The Framework MAY decide whether the information should be stored as Entities or Associates.

The data capturing and archiving mechanisms MAY resemble the following.

3.1.1.1 Type of Information: Personnel

Mechanism:

i. The Address Book is used to create Entities under the category “People” (Category names are user definable. However, a certain set of categories like People, Email etc. MAY need to always be present).

ii. User SHOULD be able to modify, add, delete or group these Entities by navigating to a virtual folder (e.g. /People/).

3.1.1.2 Type of Information: Email (Outbound)

Mechanism:

i. The email is intercepted before it is dispatched to the SMTP server.

ii. An Associate is created to contain the message body.

iii. The recipient name(s) are used to create (if required) Entities under the “People” category. The Associate is then linked to each Entity.

3.1.1.3 Type of Information: Email (Inbound)

Mechanism:

i. The email is fetched from the POP3 server.

ii. An Associate is created to contain the message body.

iii. The sender's name is used to create Entities, under the category "People". The Associate is then linked to this Entity.

3.1.1.4 Type of Information: WWW Documents

Mechanism:

i. User SHOULD be able to specify when he is to start a research project; i.e. when the web sites he visits subsequently are to be permanently stored.

ii. The project name specified by the user spawns a new Entity in the Framework.

iii. An associate is created as an archive and is linked to this Entity.

3.1.1.5 Type of Information: Textual Documents

Mechanism

i. The user after editing a document in a word-processor invokes a special GUI supplied by the Framework.

ii. The GUI allows him to draw arrows to and from (a symbol representing) the current document to any Entity creating relationships between the document and the relevant Entities.

iii. The user may optionally specify a descriptive name for the document.

iv. The document is automatically saved to disk in a manner that reflects the relationships it has to other Entities. The user may in the future fetch this document by following the relationship paths that he had created.

The GUI described in section 3.1.1.5 MAY also be made available when dealing with other types of information as well.

3.1.2 Hierarchical view of the information

The user MUST be able to view the information in the central store in a hierarchical manner that makes sense to him. Section 3.1.1 describes how this hierarchy is built up. The Framework SHOULD include a browser that is capable of displaying this hierarchy in a UI similar to that of the Windows Explorer. The browser MAY OPTIONALLY be integrated directly in to Windows Explorer, rather than being a standalone application.

This browser MAY integrate the GUI described in section 3.1.1.5 as well.

3.1.3 Archiving of information

Often, the user may want to archive portions of the information-store to removable storage media (the term storage media covers only CD-ROMs and USB Pen Drives at this point). However, this SHOULD not mean that such portions would be "forgotten" by the Framework.

Therefore, some residue MUST be left on the data store to indicate that a portion of data has been archived. For example, assume all information relating to a "Person" Entity named John is archived to a CD-ROM. When the user subsequently looks for John, the Framework SHOULD NOT declare that such an Entity was not found. Rather, it SHOULD inform the user that the Entity has been archived (and OPTIONALLY, if he wishes to proceed, to remount the relevant storage device).

3.1.4 The public API

The Framework API should expose at least the following functionality to client processes.

- Create, delete and modify Entities and Entity groups
- Create, delete and modify relationships between Entities
- List and access Entities with an API similar to that of a standard file system API.
- Execute queries for the retrieval of Entities using different criteria.

The API that exposes Entity listings MAY be designed to be similar to the `System.IO` APIs of the .NET Platform which provide directory and file listings.

3.2 Usability

3.2.1 Typical training time required

GUIs associated with the Framework SHOULD be designed such that an user with basic understanding of the Windows 2000/XP operating systems would instinctively be able to interact with them. Therefore, most GUIs MAY need to closely mimic popular Windows applications.

3.3 Reliability

This subsection specifies the following requirements associated with the reliability of the system.

3.3.1 Mean Time between Failure (MTBF)

The mean time between failures (MTBF) SHALL exceed 3 months.

3.4 Performance

3.4.1 Capacity

At this stage of the development, the Framework MAY only operate in single user mode, with no networking capabilities.

3.4.2 Resource Utilization

The Framework Shell SHOULD NOT utilize more than 40 megabytes of RAM.

3.5 Supportability

3.5.1 Coding Conventions

Any coding conventions to be used during implementation MUST be documented before the beginning of the phase, and strictly adhered to during the phase.

3.5.2 Compliance to Opens Source Standard Practices

Project ZNIX SHOULD be available to its potential users in source and binary from, and free of charge. The project MAY be hosted on a popular Open Source distribution website such that the Open Source community is aware of (and contribute to) its existence.

The website for distributing the project SHALL include provisions for the users to make comments on the product as well as to compile a wish-list of features.

3.6 Security

The Framework MUST guard against malicious scripts on remote websites from accessing its information.

The Framework MUST guard against unintended corruption of information by the user himself.

3.7 Design Constraints

No design constraint has been imposed at this stage.

3.8 Online User Documentation and Help System Requirements

The online help system should be compiled utilizing as much of non-technical terms as possible. Care SHOULD be taken not to produce an exhaustive user documentation as that would defeat the main purpose of the system as expressed in the introduction to section 2.

A comprehensive Developer's Guide MAY be compiled to facilitate 3rd party ZNIX-aware applications.

3.9 Purchased Components

N/A

3.10 Interfaces

3.10.1 User Interfaces

A GUI capable of browsing the information managed by the Framework, similar to that of the Windows Explorer SHOULD be available to the user. The GUI MUST also be capable of allowing the user to specify relationships between disparate Entities as he see fit.

3.10.2 Hardware Interfaces

The system does not require special hardware interfacing.

3.10.3 Software Interfaces

Provisions SHOULD be made so that at least the following 3rd party applications can take advantage of the Framework for the specified types of information.

Product Name	*Source*	*Version*	*Type of Information*
Outlook Express	Microsoft Corporation	6.0	Email
Outlook 2003	Microsoft Corporation	11.5	Email
Address Book	Microsoft Corporation	6.0	Personnel Information
Internet Explorer	Microsoft Corporation	6.0	WWW Documents
Opera	Opera Software ASA	7.5	WWW Documents
Firefox	Mozilla Corporation	1.0	WWW Documents
Word 2003	Microsoft Corporation	11.5	Textual Documents

Table 3.10.3-A

Targeted operating systems are Windows 2000 Professional and Windows XP.

3.10.4 Communications Interfaces

At this stage of development Project ZNIX requires no communication interfaces.

3.11 Licensing Requirements

ZNIX Framework will be licensed under GNU Library General Public License (LGPL) [WWW03].

The browser and any other tool released alongside the Framework will be licensed under GNU Public License (GPL) [WWW04].

3.12 Legal, Copyright and Other Notices

- All proprietary trademarks, product names and/or logos used on this document are trademarks of their respective owners.

3.13 Applicable Standards

All components associated with Project ZNIX should conform to the following standards:

- Windows 2000 Logo Requirements [WWW05]

The system SHALL conform to ISO 10646 (Unicode UTF-8) and OPTIONALLY ISO 10646-1 (Unicode UTF-16) standards for character set encoding. [WWW06]

Additionally, design of the system MAY yield a new set of standards that ZNIX-aware client processes MAY need to conform to.

4. Supporting Information

None specified.