# Open Document Format v1.1 Accessibility Guidelines Version 1.0

## Committee Draft

## 15 October 2007

**Specification URIs:**

**This Version:**

http://docs.oasis-open.org/office/office-accessibility/v1.0/cd01/ODF_Accessibility_Guidelines-v1.0.odt

http://docs.oasis-open.org/office/office-accessibility/v1.0/cd01/ODF_Accessibility_Guidelines-v1.0.pdf

http://docs.oasis-open.org/office/office-accessibility/v1.0/cd01/ODF_Accessibility_Guidelines-v1.0.html

**Previous Version:**

n/a

**Latest Version:**

http://docs.oasis-open.org/office/office-accessibility/v1.0/ODF_Accessibility_Guidelines-v1.0.odt

http://docs.oasis-open.org/office/office-accessibility/v1.0/ODF_Accessibility_Guidelines-v1.0.pdf

http://docs.oasis-open.org/office/office-accessibility/v1.0/ODF_Accessibility_Guidelines-v1.0.html

**Technical Committee:**

OASIS Open Document Format for Office Applications (OpenDocument) Technical Committee

**Chair:**

Michael Brauer, Sun Microsystems, Inc.

**Editors:**

Peter Korn, Sun Microsystems, Inc.
Rich Schwerdtfeger, IBM

**Related Work:**

These guidelines apply to the OASIS OpenDocument v1.1 specification, which can be found at: http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1.odt

**Abstract:**

This document is a guide for Office Applications, that support version 1.1 of the OpenDocument format, to promote and preserve accessible ODF documents. This guide is not a comprehensive guide for content mapping to platform accessibility APIs.

**Status:**

This document was last revised or approved by the OASIS Open Document Format for Office Applications (OpenDocument) Technical Committee on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/office/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/office/ipr.php.

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/office/.

This is a non-normative document.

# Notices



All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", “OpenDocument”, “ODF”, and “Open Document Format” are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the

organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

# Table of Contents

# 1 Background and Overview

## 1.1 Introduction

The Open Document Format v1.1 (ODF) is capable of encoding and persisting a tremendous amount of structural and semantic information needed by people with disabilities and the tools they use (assistive technologies) to gain access to computers and information. This document describes the things that an ODF 1.1 implementation must do in order to gain, persist, and present all of the information that ODF 1.1 is capable of persisting, so that users with disabilities are equally able to read, create, and edit documents – with full access to all of the meaning and intent – as their mainstream colleagues.

## 1.2 What is Accessibility?

The Open Document Format v1.1 (ODF) is capable of encoding and storing a lot of structural and semantic information. This information is needed by people with disabilities and the tools they use (assistive technologies), to gain access to computers and information. This document provides guidelines for ODF 1.1 implementation. A successful ODF 1.1 implementation will enable users with disabilities to read, create, and edit documents – with full access to all of the meaning and intent – just like a person without any disability.

Accessibility is about enabling people with disabilities to participate in substantial life activities that include work and the use of services, products, and information. In the context of ODF documents, this means that people with disabilities should be able to participate fully in the creation, review, and editing process of the documents. A blind person, for example, should be able to work with a document someone else created (by getting a text description of the images used). A person should be able to fill out a form without using hands. A person with poor vision should be able to read through presentation materials easily.

### 1.2.1 Types/Categories of Access

There are two or three different *types* or *categories* of *access*.

The first is *direct access*. With direct access, the person with the disability is capable of direct interaction . For example, a deaf person has direct access to a newspaper – the disability does not restrict the medium. A blind person (but not deaf) can use a phone easily.

The second category of access is *mediated access*, or access through an *assistive technology*. Here, the person with the disability is interacting with the media/medium via some other tool. A person with poor vision might use a magnifying glass to read a book (note that same person might have direct access

to a large print book). Someone who is deaf might enjoy a television show via good *captions*.

The third category of access isn't really access. A friend reading the mail to the blind recipient might fall into this category. May be we can call this as *indirect access.*

Talking about computers, direct access is what that the program itself provides to the users. The large print theme of the desktop and applications is the equivalent of the large print book to a user with poor vision. For blind users, a special program called a *screen reader* is used which mediates the user's experience. It reads the text via speech (or renders it using a *refreshable Braille display*). A user for whom the large print theme doesn't work (but still has some vision), a *screen magnifier*, which magnifies the contents of the screen, is a viable good option.

It is worth noting that many of the things we have built specifically for accessibility for people with disabilities are used and appreciated by people without disabilities. Close captioning of television broadcasts has become very popular in noisy places like bars and airports; while captioning in multimedia presentations has allowed text searches of those presentations. Similarly, the wheelchair ramps that allow people in wheelchairs to easily move from the street to the sidewalk are much appreciated by delivery personnel, parents with strollers, and anyone else on wheels (bicycles, skateboards).

### 1.2.2 Types of disabilities, and how they are addressed

There are several different types of disabilities, and they are addressed (or adapted to) with either direct or mediated access techniques. These types of disabilities mostly stem from impairments in one or more of the primary senses.

#### 1.2.2.1 Minor vision impairments

People with minor vision impairments have one or more problems that can be addressed via direct access techniques. These include some level of color blindness (roughly 10% of the male population worldwide has some level of color blindness), the inability to see text below 15 or 18 or 24 point text (an issue for most people as they age), or some reduction of their field of view that isn't very severe. To be considered a true impairment, the disability must be something that hasn't otherwise been fully addressed by corrective lenses such as glasses.

These users need the default presentation of the ODF application and ODF content rendered somewhat differently. For example, they might need a particular color scheme, or a special high-contrast scheme. They might need the size of the text and images to be larger. The specific techniques that ODF applications must employ for users with these needs are described in section 2.2.

#### 1.2.2.2 Major vision impairments

People with major vision impairments need more adaptation in the presentation of an ODF document than rendering the text in an alternate color scheme, or using a larger font. These users need the entire presentation magnified – typically between 2 and 16 times (one pixel becoming 4 to 256 pixels). Such a significant level of magnification cannot be accommodated within the boundaries of most computer screens. Instead, a second piece of software, an *assistive technology* called a *screen magnifier,* is used. It renders a magnified image to a virtual screen that is far larger than the physical screen. This *screen magnifier* software displays only the appropriate portion of the magnified screen on the physical screen: the portion that the user is interacting with at that moment of time. In addition, this *screen magnifier* may be magnifying an original, scaled image that is in a particular color scheme; or the magnifier itself may alter the colors. Some of the more sophisticated *screen magnifiers* offer additional features, such as

- reading the text they are magnifying via synthetic speech.
- panning and zooming through the text contents (so a user doesn't need to move the mouse to read and review the screen).
- extracting the ODF text contents and re-rendering it in a different font, in a specialized “text view pane” on the screen.

#### 1.2.2.3 Near or total blindness

There are people whose vision is so severe that they have effectively no usable vision at all. They use a different *assistive technology* to read and edit ODF documents – a *screen reader*. This software uses synthetic speech (or *text-to-speech*, commonly abbreviated to *TTS*) to read the contents of the screen, application, and document to the user. Such software tracks the object or text the user is interacting with (sometimes called the *locus of focus*) and reads that object or text to the user. This includes

- the letters the user is moving the text *caret* left-and-right between
- the lines of text the user is moving the *caret* up and down between
- the menu items the user is interacting with
- the items in the dialog boxes of the ODF application.

*Screen readers* are sometimes combined with *screen magnifiers*, in which case magnification also tracks this locus of focus. *Screen readers* sometimes also support *refreshable Braille displays* in which case in addition to speech, the information is rendered to the Braille display. If the user is both deaf and blind,

Braille will be the exclusive medium of information from the ODF document and application.

#### 1.2.2.4 Minor physical impairments

People with minor physical impairments have a variety of issues that prevent them from using the full keyboard or mouse easily. In order to support such users, all the functionality of the application should be operable in a non-time-dependent manner through a keyboard interface, except where the task requires analog, time-dependent input. This is also beneficial to users who don't have a physical impairment, but otherwise cannot use a mouse (such as a blind user, who cannot see the mouse pointer on the screen and so is functionally unable to effectively use the mouse).

- In case of the physical impairment, the user is allowed to use only single finger, or a *mouth stick* such that they can only make a single keystroke at a time. This is an operating system feature called *StickyKeys.* It would be employed in such a way that latches modifier keys (like Shift and Control in software,) so that the next key pressed after the latched modified key comes through to the application as if the key and the modifier were pressed simultaneously (For example; Shift then 's' to produce 'S').
- In case of the physical impairment results in hand tremors (such as Parkinson's disease), the user might hit keys accidentally en-route to pressing the key they intend to press. In such cases, another operating system feature called *SlowKeys* would be employed which rejects key presses unless they are held down for a specified amount of time.
- In case of the physical impairment results in spasms that commonly generate a second, duplicate keystroke (a 'debounce') upon release of the key, an operating system feature called *FilterKeys* would be employed which rejects duplicate keystrokes unless a specified interval of time occurs between them.

#### 1.2.2.5 Major physical impairments without speech recognition

People with major physical impairments are typically unable to control their limbs. They can perhaps shrug a shoulder, or sip and puff on a straw, or move their head or at least their eyes. But they are unable to press a single key on a traditional keyboard.

When these impairments also affect their speech (or if they simply choose not to use *speech recognition* software), such users use *on-screen keyboard* software that is connected either to a *switch device,* which they can activate (a large button mounted to their wheelchair that they can press in some fashion, or a switch embedded within a straw, which they can 'press' by sipping or puffing on that straw), or by head movement that is tracked by a camera following a

reflective dot on their forehead or glasses, or by eye movement that is tracked by a camera following their pupil.

The *on-screen keyboard* software has variety of techniques to translate these movements to the choices of individual keystrokes, or to entire words, which are then entered into the system as if they are entered from the keyboard. This is done by displaying a software keyboard on the screen, from which the user makes their choices.

Advanced *on-screen keyboard* software can actually provide remote access to accessible menus, toolbar elements, and other UI components on the software "keyboard" where the user can choose them..

Another, new type of *assistive technology* used by people with major physical impairments is the *dynamic text entry* application, which couples a predictive text system with head, eye, or switch movement to rapidly enter the most common sequences of text based on a probability model for a given language (or a given language domain such as medical terminology).

The users with major physical impairments are not able to move their body but their eyes are able to achieve more than 35 word-per-minute typing rates with such a software.

#### 1.2.2.6 Major physical impairments with speech recognition

People with major physical impairments who are able to speak clearly may use *speech recognition* software (also known as *speech to text software*, *automatic speech recognition* or *ASR*).

This *assistive technology* not only translates the users' speech to text (for entry into the ODF document), but also allows the user to control the ODF application via speech.

For example, a user of this *assistive technology* might say the utterance "Menu File, Print" in order to choose the print item of the file menu. Or they might say "Select from 'Dear Mom' to 'please send money'" in order to select a range of text in the ODF document.

#### 1.2.2.7 Hearing impairments

People with hearing impairment have difficulty in hearing clearly, or perhaps cannot hear at all. A severe impairment may not be corrected by using a hearing aid.

The users with hearing impairment use an operating system feature called *ShowSounds*. They also use a different operating system feature called

*SoundSentry*. *ShowSounds* is a system flag that indicates the application to captioned (For example; text captions displayed in an mpeg video) any speech and sounds made in a presentation. *SoundSentry* causes the operating system to generate a visual warning along with the audible warning tone.

#### 1.2.2.8 Cognitive impairments

People with cognitive impairments have one (or more) of a very wide range of disabilities. The cognitive impairments includes:

- A range of reading impairments (such as dyslexia),
- Comprehension impairments, and
- Composition impairments.

The users with cognitive impairments use one of a variety of assistive technologies, and/or application features, in order to use the computer more effectively. The assistive technologies and application features includes:

- Using a *screen reader* which reads document text via *TTS* and also highlights the words as they are spoken,
- On-line thesauruses and dictionaries with a special focus on homographs and homophones; and
- Grammar checking tools.

The users with cognitive impairments are capable of using the most popular *assistive technology* software into text-heavy applications like office suites, e-mail packages, and web browsers.

There are more severe cognitive impairments, which are not yet very well addressed - either by mainstream software, or by assistive technologies. Further research may improve the options available to these users.

## 1.3 Importance of the Accessibility of the ODF Application

Ensuring that the ODF application is accessible to people is critical for the following reasons:

1. People with disabilities together comprise one of the largest “minority” groups – nearly 20% of the worldwide population has some form of disability.
2. In many countries, supporting people with disabilities in electronic and information technology is a legal requirement for sale or use of that technology in government.

3. In many countries, providing an education to people with disabilities is a legal requirement, and many schools therefore will not purchase or use software that doesn't support people with disabilities.
4. It is the right thing to do – morally, as well as ethically.

## 1.4 Importance of Ensuring Authors Encode Accessibility Information into Documents

The ODF v1.1 specification includes many optional elements that are critical for document content accessibility.

For example;

- Optional image text and descriptions,
- Headings for tables, and
- Logical navigation order for objects in a draw layer.

It is vital that these optional elements should be used by document authors to produce an accessible document. It is therefore important that ODF applications, at the bare minimum, allow document authors to include these optional elements. But the user interface presented to the author for doing this should be straightforward, simple, easy to find, and easy to use.

The ideal ODF authoring/editing tool would have a feature or mode that audits ODF applications for its accessibility, and flags the accessibility problems it finds.

Just as all ODF applications help the author to create documents without spelling and grammatical errors, a good ODF application will likewise help the author to create documents, which are accessible to people with disabilities.

## 1.5 Putting the pieces together

Software is composed of different components which, together form an accessible reading and editing experience for people with disabilities.

The tool, which the author used to created the ODF document, must offer a solution to encode the accessibility information needed by assistive technologies.

The ODF document reader must expose ODF accessibility information to any assistive technologies running on the operating system. This information should be provided through a rich accessibility framework. with the ideal situation of it being available with the operating system.

# 2 ODF Application Accessibility

## 2.1 Keyboard Navigation Without Using a Mouse

Every function of the accessible ODF application must be possible using the keyboard. Many users with physical impairments cannot use a mouse. For example; blind users cannot use a mouse because they cannot see where it is moving, will be unable to accomplish tasks using mouse.

However, it is not enough to support a sequence of keystrokes for accomplishing every task. The sequence of keystrokes should be efficient, and should follow conventions on that operating system for doing similar tasks (for example; F10 for focusing the menu bar; ESC for canceling dialog boxes; TAB for moving between controls in a dialog box). See these URLs for the common keystroke sequences for Windows, UNIX with GNOME, and Macintosh.

Another important thing about the keyboard accessibility support in the ODF application is compatibility with Operating System keyboard support features.

For example things like *StickyKeys* and *FilterKeys*.

It is important to test the ODF application with all of these support features to avoid a compatibility problem..

Code which examines the state of modifier keys upon reception of an event, instead of the modifier flags in the event itself, is a likely source of *StickyKey* compatibility problems.

## 2.2 Theme Support (including OS fonts & colors)

ODF applications must accept the color, contrast, and font choice information, which the user has made for his/her desktop.

Users with a range of vision impairments (such as red-green color blindness) make these settings so that they can see user interfaces running on their desktop. If the desktop setting of font size is larger than standard (for example, an 18 point font) the ODF application should ensure that no text in any menu, dialog box, or other user interface element is smaller than 18 point.

Ideally the ODF application should further optionally scale the document content, linked to the desktop font size setting, without requiring additional user interaction. This extends to every part of the ODF user interface, including things like print preview.

Desktops such as the GNOME desktop on a UNIX system offer an additional settings for vision impairments custom large print and high and low contrast icons. In these cases, the accessible ODF application should work according to these options.

Desktops such as the Macintosh desktop fail to offer desktop settings for color, contrast, and font size. Ideally, accessible ODF application used on such desktops will allow the user to choose colors and fonts which meet their needs, independent of the desktop.

## 2.3 Interoperability with Assistive Technologies

One of the most important things for supporting accessibility that an ODF application must do is to be compatible with assistive technologies. For example, if the ODF application doesn't work with the *screen reader* application on a particular platform/operating system, then blind users won't be able to use that ODF application. If there is a basic level of interoperability between the ODF application and a *screen reader,* then blind users may be able to accomplish all of the reading and editing tasks, but not necessarily in a very efficient or productive manner.

The major operating systems are either shipping, or developing, an engineered accessibility framework specifically to facilitate the interoperability between assistive technologies and software applications.

The ODF application should support the accessibility framework, and fully implement its *accessibility API*, on the platform(s) it runs on. The API should fully expose the accessibility information provided in the ODF document such as *IAccessible2*.

### 2.3.1 Characteristics of Engineered Accessibility Frameworks

An engineered accessibility framework that can provide the rich support for interoperability with assistive technologies, needed to yield a productive and efficient user interface for people with disabilities, has few common characteristics:

1. An engineered accessibility framework provides a way to locate every user interface element – where it is both on the screen visually and where it is in the window hierarchy.
2. An engineered accessibility framework allows an application to describe the role (menu item, checkbox, etc.) and status (checked, selected, etc.) of every user interface element.

3. An engineered accessibility framework provides ways of conveying all the details of complex user interface elements, including:
   a. The individual characters, font, font size, font style, text style, and character bounding rectangles of text.
   b. Text editing and selection information, including the location of the text *caret,* the ability to move the location of the text *caret*, and ability to cut/copy/paste text via the accessibility interface.
   c. The ability to set the values for objects like Sliders and scroll bars.
   d. The ability to choose one of the options from the pop-up menu.
   e. The ability to locate elements within the table by their row/column and the ability to find the row/column of a given object within that table.
4. An engineered accessibility framework provides a way of conveying at least basic *relationships* between user interface elements (such as a text label labeling an edit field, and elements being members of a group such as a collection of radio buttons).
5. An engineered accessibility framework provides events or signals, indicating asynchronous actions that change any of a wide range of information related to the user interface elements, including:
   a. A user interface element appearing, disappearing, or moving
   b. A user interface element changing stated (e.g. from selected to unselected)
   c. Text being added/replaced/deleted
   d. Text changing font, font style, etc.
   e. The text *caret* moving, or the text selection changing
   f. The value of an object that takes on a range of changing values
6. An engineered accessibility framework is extensible, and as the new user interface elements are developed, an accessibility framework should be extended to communicate any new and unique aspects of such new elements.
7. An engineered accessibility framework can be implemented independent of any particular user interface toolkit or library used. Thus an ODF application is not restricted to using any particular user interface library (for example: with GNOME one needn't use *GTK+*; it is sufficient to implement *ATK* in order to be accessible in a UNIX environment).

### 2.3.2 Recommended Engineered Accessibility Frameworks

The following accessibility frameworks are recommended for ODF applications to use on their respective platforms:

- On UNIX systems, use the *GNOME Accessibility API*. This can be done by following one of several specific user interface toolkits, including *GTK+*, *UNO*, *XUL*, and *Java/Swing*; or it can be done by implementing

support for *ATK* or the *Java Accessibility API* directly, or by *AT-SPI* directly. In any case, it is highly likely that either *ATK* or *AT-SPI* support will need to be implemented for the editing/content portion of the ODF application. This is well supported by UNIX assistive technologies such as the Orca *screen reader/magnifier*, and the GNOME On-screen Keyboard.

- On the Java platform, use the *Java Accessibility API*. This can be done by using *Java/Swing* directly, or by implementing support for the *Java Accessibility API* directly. This is well supported on UNIX systems via the GNOME Accessibility framework (which bridges the *Java Accessibility API* out of the Java Virtual Machine).This is poorly supported on Microsoft Windows by assistive technologies via the Java Access Bridge for Windows.
- On the Macintosh (OS X v10.4 or later), use the Apple Accessibility API. This is supported by the built-in VoiceOver *screen reader*, and the built-in magnifier.
- On Windows XP, use the *IAccessible2* API. This is supported by JAWS and WindowEyes today, with more *assistive technology* support expected in the future.
- For web-based ODF applications, use the WAI-ARIA interface, which is supported by the Firefox 2.0 web browser on Microsoft Windows and several commercial assistive technology products including the JAWS and WindowEyes *screen readers*. Support for WAI-ARIA is anticipated on UNIX with the release of Firefox 3.0.

### 2.3.3 Supporting an Accessibility Framework

Whether the ODF application uses an existing user interface toolkit, which supports an accessibility framework, or implements all the framework support itself, it is important that the ODF application accurately exposes all of the information about all of the user interface elements that the accessibility framework can communicate. It is especially important that the ODF application fires events/signals whenever the user interface elements change the state, gain/lose focus, or their content changes.

This is critical because many assistive technologies are event driven – reacting to text appearing by speaking the text for the blind user, reacting to the *caret* moving within the text by reading the letters being moved through to the blind user, and moving the magnified view to track the *caret* for the low-vision user.

The ODF application development and testing team should test to ensure that the accessibility framework is properly implemented – both by using test tools, and also by using the ODF application through the assistive technologies available for that platform. This is ideally done by people with disabilities who are

experienced with using assistive technologies However, even just turning off the monitor for a day and using the ODF application via a *screen reader* is extremely helpful.

### 2.3.4 Dealing with the Absence of an Accessibility Frameworks

Today not every platform provides an accessibility framework that is sufficient to convey the richness of an ODF application, and which is supported by the assistive technologies on that platform. Unfortunately in these cases the only option may be to form a business relationship with one or more *assistive technology* vendors (who may then either support some non-OS-defined accessibility framework that the ODF application and *assistive technology* can agree on), or extend the reverse engineering techniques, which accessibility framework otherwise uses to provide access to that platform, or a combination of the two.

## 2.4 Special Issues for Web-based ODF Applications

For web-based ODF applications, accessibility poses special challenges. The accessibility of the ODF application is significantly dependent upon the web client being used.

Web-based ODF applications should utilize the W3C WAI-ARIA specification (currently in draft form) for conveying all dynamic/interactive portions of their interface. Use of WAI-ARIA - through a browser that supports WAI-ARIA, such as Firefox, provides support for assistive technologies by conveying the content, meaning, context, and updates/changes of the user interface.

Further, the web-based ODF application must accept the CSS settings set by the users, in order to support color, contrast, and font settings, which the users indicate either on their desktop, in their web client, or via their own custom CSS.

The best choice of an accessible web client for dynamic web applications on a desktop computer (as of the day of release of this document) is Firefox on Windows. Firefox works to support exposure of WAI-ARIA accessibility API information via Firefox on UNIX and Macintosh systems is underway. The UNIX work is expected to finish in Firefox 3.0 by mid-2007.

## 2.5 ODF Help System Accessibility

Help systems are often separate application from the ODF application. The Help system provides help about the ODF application, which from the end-user point of view, is a feature of the ODF application itself.

An ODF application is not fully accessible unless the associated help system is also accessible.

# 3 Document Content Accessibility Basics

Access to content for people with disabilities has a number of aspects. These are discussed below.

## 3.1 Document Navigation or Structure

Well written documents generally have a recognizable structure. An example structure is shown below

| | |
|---|---|
| Title | Tells the user about the contents in the documents. |
| Summary | Tells the user the important points covered in the document. |
| Table of content or similar navigation aid | Enables the user to find their way around the document. |

From this structure the reader can determine if the document is of interest and provides the navigation aid which may be used to quickly access specific content.

Access to the parts of a document that provide these clues is key to navigation.

A title is visible title text marked in XML as a text:h element. It is important that implementations support the user in generating appropriate XML. Visually bold centered text is not the same as a title as far as XML is concerned.

The appropriate use of styles greatly helps with accessibility. Use named styles in your document; these provide the foundation of the document structure which is used by access technology to convey meaning to the user.

### 3.1.1 Text document structure

The text document structure contains

- Styles,
- Structure and
- Visual presentation.

Named styles (heading 1, Default, List indent etc) are available to the user and provide structure in the saved XML file on disk. Additionally each of these named

styles may be formatted for a given visual presentation. It is important to realize that the appropriate use of styles helps greatly, with accessibility providing the foundation of the document structure.

Authors should use named styles within a document since this provides the foundation for the document structure. With regard to visual presentation, this also allows a single style change to be reflected throughout the document. Style names can also be used when creating a table of contents. If two visually identical paragraphs are created, one using named styles and the other using styling attributes, it is important to realize that the XML produced will be different for each case.

The structure of an ODF text document, refers to the use of named styles. The automated processing of style information is used to generate a table of contents, cross references and more importantly, present the overall structure of the document for the reader.

When the document is saved into it's default form, an ODF document provides a structure based on the named styles which allows navigation and access for other tools, using the Extensible Markup Language (XML) semantic markup.

Hence the structure is important for more than one reason.

Other XML tools can be used to abstract a quick overview if needed, by selecting specific named styles which are key to the document structure.

Implementers need to assist the user in using named styles as opposed to modified font sizes and other presentational aspects.

There are three possible ways to obtain entries in a table of contents. These are, the headings (heading level 1 etc.), table of contents index marks, and named paragraph styles. The table of contents may be used as a document overview, so it is important from the point of view of accessibility as it helps readers navigate a document.

Advice: Use named styles in your document; these provide the foundation of the document structure which is used by access technology to convey meaning to the user.

### 3.1.2 Presentation document structure

For a slide presentation similar logic may be used. The title of each slide presents a variation of a table of contents, providing the person using access technology with an overview of the presentation content. With meaningful titles a presentation outline is available even if the body of each slide is fully graphical. Beyond this, it is up to the author to help a user comprehend the presentation structure.

Within a single slide, ODF has the idea of a logical navigation order (see ODF 1.1, section 9.1.4). This enables the author to specify how the slide should be *read.* If the implementor helps the author to specify this navigation order then the end user can readily navigate between the objects on a slide as intended. For example, a keyboard user may use the TAB key to skip between the images on a slide, and access technology, which can present information about the image based on the textual alternative description.

Advice: Ensure a meaningful title for each slide. Ensure sufficient text on a single slide to extract essential information. Also ensure that there is a meaningful TAB navigation order for items within the slide.

### 3.1.3 Spreadsheet document structure

There is little structure to inform the reader about the sheet, unless a spreadsheet has multiple sheets.

An informative heading given to each sheet can provide on overview of the contents of a spreadsheet. Implementors can help by making it easy to add such headers.

Row and column headings provide an anchor for the reader when navigating through the spreadsheet. If the data is accessed serially, as in the case where someone is accessing the spreadsheet via a text to speech system or a braille system, then it is easy to forget the heading of the column being accessed. The implementor can help by making it easy for the user to add row and column headings, and ensure that these are correctly marked up in the saved XML.

Advice: A reader normally looks for headings at the top of a spreadsheet. Column and row headings are essential for access technology.

### 3.1.4 List structure

If not used properly lists can confuse the reader, especially if they are not well structured and presented. Nested lists are are often visually indented from the parent list item. This is not the correct way to produce well structured lists. Implementors should provide an appropriate nested structure for nested lists and help users create lists in an appropriate way.

Advice: Use named styles in your document; these provide the foundation of the document structure which is used by access technology to convey meaning to the user. This is particularly important in nested lists.

### 3.1.5 Page breaks

When print users talk about printed matter, a common frame of reference is to use page 34, third paragraph from the bottom, or similar references. Unless a reader with sight problems can access the page numbers, this frame of reference is clearly an exclusion. Implementors should provide notification of page breaks on export.

The visual page breaks used in the print layout mode should be made available to access technology.

### 3.1.6 Table navigation

Table navigation is especially important for tables which span multiple pages.

It is necessary to indicate clearly which cells contain row and column headings. Implementers can help by prompting users to identify headers and marking them semantically within table:header-row. This allows access technology to inform the user what the column header is for the cell having current focus.

Users should ensure that those cells are structurally marked (using named styles) rather than visually marked (emphasis etc.). Implementers should ensure that table header content is available to access technology no matter what API is in use. The reason is to ensure that users have access to header information even when there are a number of pages which make up the table.

Advice: Use styles to name the row and column headings appropriately. Repeat headings if the table spans more than one page.

#### 3.1.6.1 Tables within presentations

ODF 1.1 does not support structural tables within presentations. Implementers can help by using embedded spreadsheet tables, which do have a structure. This is another case where visual and structural information can differ. Embedded spreadsheet tables have a fully navigable structure.

## 3.2 Provision of alternative text versions of document content.

When a blind user reads your document they will not have access to any diagrams or images included within it. For example, attached to this paragraph is an image. What can you find out about it? For a reader using access technology, the provision of an alternative form is essential if they are to gain access to the information, which is what the author generally wants. The implementor can help by making it easy for the author to add an alternative (generally a textual alternative) to the image.

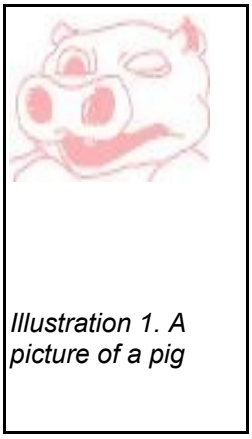

*Illustration 1. A picture of a pig*

The implementor can also help by providing a function to add an alternative text representation in a consistent manner. If the user has to select different drop-down menus, with different fly-out options in each application, then interest is soon lost and the reader loses out. Choose a method which is common across applications and which helps the author add content easily. If the method works with line drawings, embedded images, audio clips etc., then the author gains confidence in the application and can help the reader gain access to the information provided by the author.

The image may be described in one of two ways. A simple decorative image can be described using the `<svg:title>` element. For a more complex image (or diagram etc.) there is the `<svg:desc>` element (equivalent to the HTML `longdesc` element) which provides a way of entering a more complete alternative to the visual object.

Next there is the `draw:caption` element. Implementors can help by encouraging the addition of *captions* (often a few words providing a label for the object which are visually associated with the object). *Captions* are associated with the description by means of the `draw:caption-id` attribute. This must be added to link the object and its description.

Implementors must provide the means to add these descriptions in a consistent manner across all office applications.

Advice: Encourage users to add textual descriptions for any images which are important to ensure the documents information is made available to all readers.

### 3.2.1 Image maps

It is possible to assign areas of an image as being active with a hyper-link to another URI. This is commonly known as an imagemap. As with any image, a brief description should be provided using the `<svg:title>` element, or a longer description using `<svg:desc>` if a longer text alternative is needed.

These text descriptions should provide information about the content at the end of any links. Implementers can help the author to enter this by providing a standard dialog, letting the author enter text for the alternative descriptive text.

### 3.2.2 Audio inserts

When a reader with a hearing impairment receives one of your documents will he or she hear what you have inserted? For example, if an audio object were attached to a paragraph, would you know about

it? What do you know about it? What volume will it play at? How can a reader with a hearing impairment adjust the playback volume? If a reader has insufficient auditory sensitivity to access the clip, how can the implementor help the author to provide an alternative presentation? The most general method is to provide a textual alternative. How easy is it for an author to add that? Must the user learn a new technique? Help the reader by helping the author.

### 3.2.3 Complex diagrams

If you enter a complex diagram like an organization chart as part of a report (See Figure 1), do you rely on readers having full access to it? What if one of them couldn't see it or couldn't make out the detail? Does this mean it's of no interest to them? No!

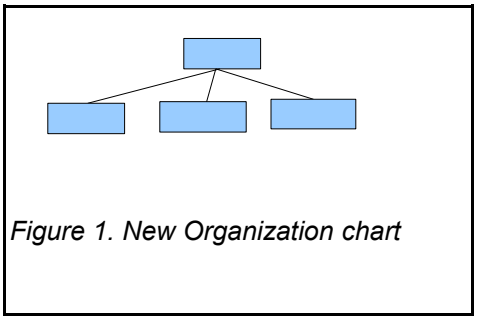

*Figure 1. New Organization chart*

This kind of consideration matters if the contained information is important to the comprehension of the entire document. How can the implementor help the writer of the document? Consider the range of disabilities for which this information may be presented accessibly and don't make assumptions about your reader, you may be surprised.

An implementer should ensure a consistent way in which the author can add a simple textual alternative. To the reader, the graphic is a variant form of an image, but to the author, the alternative text is another way of presenting the information. Implementors should help the user by making it easy to provide an alternative solutions.

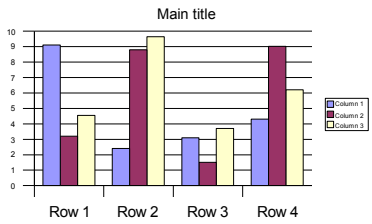


*Illustration 1: Sales Graph, 1998 to 1999*

Business reports often include graphs. I've included a dummy sales graph linked to this paragraph. It may be important to your report, it could be a crucial element of it. If the reader can't get at the information then the authors efforts are of little use. The reader needs to be aware that the graphic is inserted in the document and need to have access to it.

For a complex graph the textual alternative could be equally complex. A good implementation will provide facilities which enable alternative text from a few words to a 100 word description. Does the input area flex with use? An implementor should try using it to describe a couple of graphics seen in text documents.

Advice: Ensure that readers who can't access your non-text content have an alternate textual description of the information you provide in a graphical or audio modality.

### 3.2.4 Summary

Each of these (and there will be more) shows how we individually access information. Many of us scan information visually. Others don't, or can't. If an author provides equal access to all readers they can feel confident in reaching a maximum possible audience. If an author writes without consideration for the audience they may totally miss an important part of the information.

If an implementor makes it easy for an author to consider and meet the needs of the audience then the objective of transmitting information is achieved.

A subsequent section of the document may depend on a graphic to understand a later section. If a reader has totally missed that graphic (for whatever reason), then the document has less utility.

If you know all the recipients of your document then consider them all. If you are writing for a general audience then this document will provide guidance on how you can make it easier for them to access the information that you are passing on.

### 3.2.5 Why do anything about it?

Consider receiving a document where one third of it is blacked out, or blank. Even if the summary makes sense, you are left wondering what you have missed. The cause for such a document may be censorship or a careless writer. If you are unable to access visual, auditory and textual content in a document then you are in the same situation when you receive a document prepared without consideration for accessibility.

### 3.2.6 Summary, graphical and audio objects

For each change in media, think of potential readers who may not be able to access it. If the information is important, provide an alternative format for the reader.

## 3.3 Special considerations for subtables

It is recommended that `<table:is-subtable>` should not be used. The ODF specification for `<table:is-subtable>` requires the use of a special cell addressing scheme. Cell addresses are an important structural aid for users who are blind and not able to visually determine the location of a cell within its table. Since the subtable addressing scheme is different from the normal one it is

disorienting. The same visual effect can be obtained using `<table:number-rows-spanned>` and `<table:number-columns-spanned>`. The use of these attributes will result in cell addressing which is consistent with the surrounding table.

The usage of sub-tables can be easily be disorienting as described below:

| ***A1*** | ***B1*** | ***C1*** |
|---|---|---|
| A2 | .B2.A1 | .B2.B1 |
| | .B2.A2 | |

*Table 1: subtable example*

In this example each cell contains the cell address as defined by the ODF specification. Examples of cell address are .B2.B1 and C1. Cell contents and cell addresses are provided to users of *assistive technology* (AT). Cell addresses provide essential location information to AT users.

In Table 1, above, there is a B2 (.B2.B1) under an C1. This location information will be confusing and disorienting for the user with visibility impairment. Without significant extra effort, a blind user has limited access to the subtable.

The cell location information is also important when using tables with row and column headers, for example the first row and first column in the table above. In this table the use of `<table:is-subtable>` results in a cell addressing scheme that is disjoint from the addressing scheme of the row and column headers and the semantic relationship is broken. For example, in the prior table cell C1 is a header of a B2 cell.

The visual equivalent of the prior table is the following table which will result when using `<table:number-rows-spanned>` and `<table:number-columns-spanned>`. In this case cell C1 is the header of cell C2 and a meaningful location relationship between the two cells is maintained.

| ***A1*** | ***B1*** | ***C1*** |
|---|---|---|
| A2 | **B2** | **C2** |
| | **B3** | |

*Table 2: Table using number-rows-spanned and `number-columns-spanned` attributes*

# 4 Converting between ODF and other Document Formats

Accessibility issues surrounding document format conversion are often overlooked. This section covers issues that an office application must conserve to preserve accessibility when converting between ODF documents and other Office documents such as HTML, *DAISY*, or MS Office.

## 4.1 Preservation of Alternative Text

Alternative text is used to provide alternatives to non-text objects such as drawing or images. Throughout the ODF specification, `<svg:title>` and `<svg:desc>` are used to represent the short accessible name and long description of these document elements. Long descriptions are not new to HTML however they are new to office documents formats and consequently long descriptions may not need mapping.

Alternative text is used to describe a graphic or drawing object. The users with visibility impairment prefer to hear a short description for these document elements when navigating a document with the *screen reader* as it improves productivity as compared to using a long description.

The ODF Accessibility Subcommittee introduced long descriptions for drawing objects to provide more information about the drawings. This is particularly important for complex drawings formed from a group of smaller shapes.

When converting to and from ODF it is essential that alternative text is preserved.

### 4.1.1 <svg:title>

The `<svg:title>` attribute represents the short accessible name of the non text drawing object.

Applications, supporting ODF, must ensure that the double quotes come through in the submitted document.

| *ODF 1.1 Element* | *MS Office* | **HTML element using alt= “..”** | *HTML element using title=“..”* |
|---|---|---|---|
| `<draw:rect>` `<draw:line>` | [1]Microsoft Office format | `<IMG` | This is not used for the |

| *ODF 1.1 Element* | *MS Office* | **HTML element using alt= “..”** | *HTML element using title=“..”* |
|---|---|---|---|
| <draw:polyline><br><draw:polygon><br><draw:regular-polygon><br><draw:path><br><draw:circle><br><draw:ellipse><br><draw:g><br><draw:page-thumbnail><br><draw:frame><br><draw:measure><br><draw:caption><br><draw:connector><br><draw:control><br><dr3d:scene><br><draw-custom-shape> | mapping is incomplete due to limited alternative text support. | alt=””> | short name for images. |
| <text:a> | Hint text for links. The actual format is not available for listing here. | No alternative text attributed is provided. | <a title=””> |

1 Microsoft Office does not support short names and long descriptions and therefore authors use alternative text as either short names or long descriptions. When converting from ODF to Microsoft Office .doc, .ppt, or .xls formats, office applications should first map the short name to the alternative text, if it exists. If not, the long description should be mapped. When converting from Microsoft Office .doc, .ppt, or .xls formats to ODF long descriptions, in the absence of a short name, assistive technologies should process the long descriptions as the alternative text for a drawing object. It should also be noted that since the Microsoft formats are not open our analysis of the accessibility information provided by Microsoft Office was taken from their Document Object Model APIs. These APIs form the interoperability layer with assistive technologies on Windows. Consequently, we do not have actual .doc format information listed here.

| *ODF 1.1 Element* | *MS Office* | **HTML element using alt= “..”** | *HTML element using title=“..”* |
|---|---|---|---|
| <draw:layer> | No mapping | N/A: No drawing layer in HTML. | N/A: No drawing layer in HTML. |
| <draw:image> | | <IMG alt=””> | This is not used for the short name for images. |
| <draw:area-rectangle> | No mapping | <AREA Shape="rect" alt= “...”> | This is not used for the short name for images. |
| <draw:area-circle> | No mapping | <AREA Shape="circle" alt= “..”> | This is not used for the short name for images. |
| <draw:area-polygon> | No mapping | <AREA Shape="poly" alt= “..”> | This is not used for the short name for images. |

*Table 1* illustrates the proper mapping of ODF accessibility short names to HTML 4.01 and Microsoft Office .doc format.

### 4.1.2 <svg:desc>

The <svg:desc> element represents the long description of a drawing object.

| *ODF 1.1 Element* | *MS Office* | *HTML element using alt= “..”* | *HTML element using title= “..”* | *HML element using longdec* |
|---|---|---|---|---|
| <draw:rect> | [2]Microsoft | Not | Use <img | Use: |

| *ODF 1.1 Element* | *MS Office* | *HTML element using alt= “..”* | *HTML element using title= “..”* | *HML element using longdec* |
|---|---|---|---|---|
| <draw:line><br><draw:polyline><br><draw:polygon><br><draw:regular-polygon><br><draw:path><br><draw:circle><br><draw:ellipse><br><draw:g><br><draw:page-thumbnail><br><draw:frame><br><draw:measure><br><draw:caption><br><draw:connector><br><draw:control><br><dr3d:scene><br><draw-custom-shape> | Office format mapping is incomplete due to limited alternative text support. | applicable to the long description. | title= “;;”> if this is a simple string. | <img longdesc=”” ><br><br>if this is more than a simple text string such as it having embedded links, etc. |
| <text:a> | No mapping | Not applicable to the long description. | <a title= “..”> | Not supported in HTML <a> element |
| <draw:layer> | No mapping | Not | <img title= “..”> | Not |

2 Microsoft Office does not support short names and long descriptions and therefore authors use alternative text as either short names or long descriptions. When converting from ODF to Microsoft Office .doc, .ppt, or .xls formats, office applications should first map the short name to the alternative text, if it exists. If not, the long description should be mapped. When converting from Microsoft Office .doc, .ppt, or .xls formats to ODF long descriptions, in the absence of a short name, assistive technologies should process the long descriptions as the alternative text for a drawing object. It should also be noted that since the Microsoft formats are not open our analysis of the accessibility information provided by Microsoft Office was taken from their Document Object Model APIs. These APIs form the interoperability layer with assistive technologies on Windows. Consequently, we do not have actual .doc format information listed here.

| *ODF 1.1 Element* | *MS Office* | *HTML element using alt= “..”* | *HTML element using title= “..”* | *HML element using longdec* |
|---|---|---|---|---|
| | | applicable to the long description . | | Supported |
| <draw:image> | [3]Microsoft Office format mapping is incomplete due to limited alternative text support. | Not applicable to the long description . | <img title= “..”> | Use: <img longdesc= ””> if this is more than a simple text string such as it having embedded links, etc. |

*Table 2* illustrates the proper mapping of ODF accessibility long descriptions to HTML 4.01 and Microsoft Office.

HTML's Longdesc is a special case situation. It should only be addressed when importing from an *HTML* document. In the case where longdesc refers to a separate file, the file may contain a significant amount of HTML content.

3 Microsoft Office does not support short names and long descriptions and therefore authors use alternative text as either short names or long descriptions. When converting from ODF to Microsoft Office .doc, .ppt, or .xls formats, office applications should first map the short name to the alternative text, if it exists. If not, the long description should be mapped. When converting from Microsoft Office .doc, .ppt, or .xls formats to ODF long descriptions, in the absence of a short name, assistive technologies should process the long descriptions as the alternative text for a drawing object. It should also be noted that since the Microsoft formats are not open our analysis of the accessibility information provided by Microsoft Office was taken from their Document Object Model APIs. These APIs form the interoperability layer with assistive technologies on Windows. Consequently, we do not have actual .doc format information listed here.

The *user agent* would need to decide, whether to prefer to consume the contents of the file and store it in a test string. Typically, the title text would suffice. Very few web page authors make use of longdesc as it requires significant amount of work to create the additional file. Use of the title attribute to store significant amounts of text would be easier to do.

## 4.2 Preservation of Document Structure Hierarchy and Landmarks

The preservation of document structure is critical for accessibility. Structural information is used to give the person with a disability context of where they are within a document. The following structural information must be preserved when converting between document formats:

- Heading elements including their level (perhaps depth? Or hierarchical level)
- List structural elements
- Footer elements
- Speaker note elements
- Tables
- Page breaks and page numbering

  The preservation of this information must be done by ensuring corresponding tags in the existing content model are mapped during format conversion. Use of styling is inadequate. Content, and not styling, is mapped to platform accessibility application interfaces (APIs) to convey structure to assistive technologies.

### 4.2.1 Preservation of Heading Structure

ODF supports headings and heading levels. It is important that the headings and its levels should be preserved to maintain the structural semantics.

As an example, we will cover the conversion of ODF heading and heading levels to HTML. Any `<text:h>` element which does not provide a level, in the form of the text:outline-level> attribute is considered a level one header. HTML includes elements pertaining to H1, H2, H3, H4, H5, H6.

Office applications should consider limiting their heading levels to the document formats the support exporting to.

Ignoring the styles, the following are the header content:

```
<text:h text:style-name="Heading_20_1" text:outline-level="1">Heading 1</text:h>
<text:p text:style-name="Text_20_body">Sample text</text:p>
```

```
<text:h text:style-name="Heading_20_2" text:outline-level="2">Heading 1.1</text:h>
<text:p text:style-name="Text_20_body">More sample text</text:p>
<text:h text:style-name="Heading_20_3" text:outline-level="3">Heading 1.1.1</text:h>
<text:h text:style-name="Heading_20_1" text:outline-level="1">Heading 2</text:h>
```

which converts the above information in HTML format as shown below.:

```
<H1 CLASS="western">Heading 1</H1>
<P>Sample text</P>
<H2 CLASS="western">Heading 1.1</H2>
<P>More sample text</P>
<H3 CLASS="western">Heading 1.1.1</H3>
<H1 CLASS="western">Heading 2</H1>
```

Although levels in ODF headers are defined by attributes, the conversion is straight forward, given the predefined heading levels in HTML. Microsoft Office .doc format also supports heading levels and the conversion should also be straight forward. Microsoft Office .doc format is proprietary and therefore the actual conversion is not illustrated here. By preserving the heading levels in the content, an office application can map the information to standard *accessibility APIs* to be used in an alternative form such as a *screen reader* would be using speech application.

### 4.2.2 Preservation of list Structure

When converting to and from ODF it is important that the list's structural hierarchy is preserved. *Assistive technology* can use this information to convey contextual list with detail information. The list structure accessibility is lost when an office application uses styling to perform the indentation.

As an example the Figure 1 shows a conversion from ODF list structure to HTML list structure. Microsoft Office also supports preservation of list structure. The actual format is not available for listing here.

Figure 1: List Structure Preservation – ODF to HTML 4.0.1

Below is the example of the ODF Content:

```
<text:list text:style-name="L1">
      <text:list-item>
         <text:p text:style-name="P1">Hello</text:p>
           <text:list>
             <text:list-item>
```

```
                  <text:p text:style-name="P1">Hola</text:p>
                    <text:list>
                       <text:list-item>
                             <text:p text:style-name="P1">Hola ODF
expert</text:p>
                       </text:list-item>
                  </text:list>
            </text:list-item>
          </text:list>
    </text:list-item>
     <text:list-item>
          <text:p text:style-name="P1">Goodbye</text:p>
            <text:list>
             <text:list-item>
                 <text:p text:style-name="P1">Adios</text:p>
             </text:list-item>
           </text:list>
     </text:list-item>
     <text:list-item>
         <text:p text:style-name="P1"> Welcome </text:p>
     </text:list-item>
</text:list>

<text:list text:style-name="L2">
    <text:list-item>
        <text:p text:style-name="P2">Step 1</text:p>
        <text:list>
            <text:list-item>
                  <text:p text:style-name="P2">Step 1.1</text:p>
            </text:list-item>
            <text:list-item>
                  <text:p text:style-name="P2">Step 1.2</text:p>
            </text:list-item>
        </text:list>
    </text:list-item>
    <text:list-item>
         <text:p text:style-name="P2">Step 2<</text:p>
    </text:list-item>
    <text:list-item>
         <text:p text:style-name="P2">Setp 3</text:p>
     </text:list-item>
```

```
</text:list>
```

Below is the converted HTML content (without styling):

```
<ul>
     <li>Hello
        <ul>
            <li>Hola
                <ul>
                    <li>Hola ODF expert</li>
                </ul>
             </li>
        </ul>
     </li>
     <li>Goodbye
        <ul>
           <li>Adios</li>
        </ul>
     </li>
     <li>Welcome</li>
</ul>
<ol>
     <li>Step 1
        <ol>
             <li>Step 1.1</li>
             <li>Step 1.2>>
        </ol>
     </li>
     <li>Step 2</li>
     <li>Setp 3</li>
</ol>
```

To properly preserve content structure, each embedded list must be enclosed within the enclosing list item as it has been done in ODF.

In this example an *assistive technology* can accurately convey the property list depth information as well as the number of siblings at a given level.

### 4.2.3 Preservation of Page Header and Footer structure

The preservation of page header and footer information in a page, when supported between file formats, is essential to maintain the structural context and to allow the user to navigate within the sections of a page.

In ODF, this information is stored in the styles.xml file within the master-styles section as shown in this example:

```
<office:master-styles>
   <style:master-page style:name="Standard" style:page-layout-
name="pm1">
      <style:header>
         <text:p text:style-name="Header">This is a header</text:p>
     </style:header>
     <style:footer>
        <text:p text:style-name="Footer">This is a footer</text:p>
    </style:footer>
   </style:master-page>
</office:master-styles>
```

Neither HTML nor XHTML supports a page header or footer construct. However, a new role attribute has been created by the HTML working group which provides standard roles for standard web page *landmarks* as part of its new XHTML Role Attribute module. The standard landmark of content may be used to identify the role header and footer for a page. Microsoft Office supports preservation of header and footer structure. The actual format is not available for listing here.

### 4.2.4 Preservation of Speaker Note elements

When converting from one presentation format to another, the office application must preserve a clear outline of the speaker note. If speaker note sections are only identified using styling, in *assistive technology*, the user may not be able to identify the contents of the speaker notes.

In short, speaker notes should be considered a structural landmark which means that it must be possible for an *assistive technology* to navigate to the speaker note and access the contents of the speaker note..

The following example denotes an ODF speaker notes section using the presentation:notes element.

```
 <presentation:notes draw:style-name="dp2">
  <office:forms form:automatic-focus="false" form:apply-design-
mode="false" />
```

```
  <draw:page-thumbnail draw:style-name="gr1" draw:layer="layout"
svg:width="12.768cm" svg:height="9.576cm" svg:x="4.411cm"
svg:y="2.794cm" draw:page-number="1" presentation:class="page" />
  <draw:frame presentation:style-name="pr3" draw:text-style-name="P3"
draw:layer="layout" svg:width="15.021cm" svg:height="10.63cm"
svg:x="3.292cm" svg:y="13.299cm" presentation:class="notes">
  <draw:text-box>
  <text:p text:style-name="P3">This December, 1991 Byte article was written by Rich
Schwerdtfeger with side bars by Joseph Lazzaro</text:p>
  </draw:text-box>
  </draw:frame>
</presentation:notes>
```

When converting to HTML, there is no element which would denote a speaker note. However, a new role attribute has been created by the HTML working group which provides standard roles for standard web page *landmarks* as part of its new XHTML Role Attribute module. The standard landmark of note may be used to identify the role header and footer for a page. Rendering of the speaker note could be performed through the use of script which changes the visibility of the speaker note section through the use of a Cascading Style Sheet.

Microsoft Office supports preservation of header and footer structure. The actual format is not available for listing here.

### 4.2.5 Preservation of Table structure

Preservation of table structure is essential for a disabled user to navigate a table. A user with visibility impairment cannot perceive where they are in the table without preserving context.

The following must be preserved when transforming one document format to the other:

- Row Header information
- Column header information (if supported)
- Row demarcation elements
- Cell demarcation elements
- *Captions* (if supported) – may need to move this to alternative text section
- Spreadsheet Equations must have cell references consistent with table structure

ODF 1.1 does not support native tables in presentations. Users importing non-ODF slides that contain tables need access to the table structure via their *assistive technology*. Therefore tables imported into an ODF application from

another file format must have their structure preserved, and when saved as ODF files should be saved as embedded spreadsheets. The ODF Technical Committee is reviewing adding native table support in a future version of ODF.

Microsoft Office .doc and .ppt both support table structure with the exception of heading information. Conversion from ODF to corresponding Microsoft Office formats should be straight forward and all structural information must be preserved with the exception of table header that are not supported by Microsoft Office. The actual Microsoft office formats are not available and therefore the corresponding Office elements are not listed here.

### 4.2.6 Preservation of Page Breaks

A visibility impaired user or mobility impaired user may wish to navigate by page as this reduces the number of keystrokes required to navigate a document.

In fact, ODF 1.1 has introduced a soft page break. When combined with hard page breaks (inserted by the user and not through automatic pagination) the two items in combination allow for page-based navigation in a *DAISY* digital talking book conversion. *DAISY* readers are used by blind and partially-sighted people.

## 4.3 Maintaining the accessibility of Form Elements

The following accessibility features of form elements must be maintained

- label associations within form controls
- form control titles

ODF borrows these concepts from HTML forms. ODF supports the XForms model, but not XForms controls. In XForms, label would be a child of the form, which controls it labels.

In ODF, a label references is a form control, like HTML. The labeling is done by using the form:for attribute and using the form:id of the form control. The following example shows the use of label and title as applied to a date field.

```
<form:textarea form:name="TextBox" form:control-
implementation="ooo:com.sun.star.form.component.TextField"
form:id="control1" form:convert-empty-to-null="true" form:title=‿Enter
your Date of Birth‿>
  <form:properties>
   <form:property form:property-name="DefaultControl" office:value-
type="string" office:string- value="com.sun.star.form.control.TextField"
/>
  <form:property form:property-name="MultiLine" office:value-
type="boolean" office:boolean-value="true" />
  </form:properties>
```

```
</form:textarea>
  <form:fixed-text form:name="LabelField1" form:control-
implementation="ooo:com.sun.star.form.component.FixedText"
form:id="control2" form:label="Date" form:for="control1" />
```

When exporting to HTML, the label and its association to the form field, as well as the title, should be preserved as follows (note this is without full styling):

```
<FORM NAME="Standard">
  <LABEL for="control1">Date</LABEL>
  <P STYLE="margin-bottom: 0in">
     <TEXTAREA NAME="TextBox" ROWS=3 COLS=29 WRAP=SOFT STYLE="float:
left; width: 2.48in; height: 0.65in id=control1" title‿Enter your Date
of Birth‿></TEXTAREA>
     <BR>
  </P>
</FORM>
```

Microsoft Office .doc format does not support forms.

## 4.4 Maintaining Association Captions

ODF allows author to create a *caption* for drawing objects. Structurally, it is not clear that the *caption* is associated with the drawing object. To address the problem a `draw:caption-id` was created to establish a relation between the drawing object and the text *caption* describing the object. The id refers to the XML:id of the prose describing the drawing object.

When converting to other document formats it is important to maintain the relationship between the *caption* and the corresponding drawing object as long a a mechanism is provided.

When converting to HTML, drawings are converted to images. Although HTML, natively, does not provide a *caption* relationship between the image and the corresponding prose, but it does provide another association called the long description or longdesc. In this example a code is written, which links longdesc to a paragraph containing a unique id defining the prose text. Additionally, an HTML anchor containing Figure 1.0, from the start of the text, links backward to the named anchor representing the start of the image. By maintaining the relationship between the image and its *caption* it makes it easy for the user to find the *caption* text for a given drawing:

```
<a id="fig1" name="fig1" />
```

```
<img width="640" height="512" alt="Accessibility information mapped to a
DOM element in the DOM" src="accessibleelement" longdesc="#desc_fig1"/>
<br />
Figure 1.0 Accessibility Interoperability at a DOM Node without JavaScript</p>
<p id="desc_fig1">Long description of the figure....
<a href="#fig1">Figure 1.0</a>
```

It is important to note that if other document formats do not have provision for addressing *captions* then exporting ODF to that document format will result in a reduction in accessibility. Microsoft Office .doc format does support *captions* for tables and drawings, however, the actual .doc format is unavailable and therefore it is unclear if the *caption* can implicitly be paired with the document element being captioned.

## 4.5 Preservation of MathML accessibility information

MathML is the accepted standard for representing mathematical notation in XML. Accessibility is an important goal of MathML. It can be used by synthetic speech applications or translated to Braille math codes for use with an embosser to print hard copy Braille or on a *refreshable Braille display* connected to a computer.

MathML supports navigation within a mathematical expression. MathML also provides support for synchronized highlighting of what is spoken for the math, which is an important accessibility, considering both for users with low vision as well as people with-print related learning disabilities, such as dyslexia.

For these reasons, MathML is being incorporated into other accessibility standards, such as *DAISY* and other ebook and document standards. MathML is also an accepted standard for compatibility between computer applications. Two of the many applications that can accept and/or produce MathML are Microsoft Office 2007 and Mathematica.

MathML is supported in XHTML by Firefox, or Internet Explorer when used with the (free) MathPlayer plug-in, which interfaces with *screen readers* to make the math accessible. MathML is also supported in HTML by Internet Explorer and in an experimental version of Firefox, but due to the lack of universal MathML implementation in all browsers, HTML cross-platform compatibility is difficult to ensure.

An img tag with alt text does provide cross platform compatibility, but at a cost of all but the most rudimentary level of accessibility to math. Math images cannot be enlarged well for large print, nor can their colors be changed for those with color disabilities.

Math equations supplied in image formats cannot be navigated (other than the description supplied in the alt text). Maths equations cannot be translated to Braille, nor can support synchronized highlighting

It is also important to note that generating good alt text for math is not simple because it must be unambiguous. For example, "1 over n plus 2" can mean "1/n+2" or "1/(n+2)". Furthermore, certain categories of readers may require enhanced verbosity while other may find this more confusing.

Although images with alt text might seem like an accessible alternative, their use should be discouraged for the reasons listed above, and MathML should be used instead of images whenever possible.

The Microsoft Office specification for its (.doc, .ppt, .xls) math support is unavailable and therefore it is not known how to preserve math accessibility information when converting between ODF and Microsoft office formats.

## 4.6 Preservation of Synchronized Media (animations) SMIL

Animations in ODF applications are achieved using the *Synchronized Media Integration Language* or SMIL which supports accessibility. SMIL supports accessibility by allowing synchronization of text with audio for significant audio formats. ODF only uses SMIL to synchronize animations such as fly ins. For this, no additional work is required as long as when content is rendered the user continues to maintain focus and the user continues to control the animation.

# 5 Special Consideration for alternative media produced from ODF

Just as ODF documents are printed, they may be printed in large-print, braille, or produced according to specialized multimedia standards for the use of persons with disabilities. It may seem counter intuitive, but audio is a primary medium for providing accessibility to many persons with disabilities. Whether it is an audio recording of someone reading text, or a real time computer generated "*Text-To-Speech (TTS)*" rendition, persons who are blind, who live with severely impaired vision or with learning disabilities, often use audio as their primary reading modality.

While we do not expect ODF applications should become audio recording and editing applications, there are critical considerations that should be observed in order that ODF documents might be easily used to create usable alternative media, including audio, braille and large print.

## 5.1 Where and How Audio is Used for Accessibility

Audio renditions of textual content were commonly used to provide alternative access for print disabled individuals long before "audiobooks" became commercially available. When large print versions cannot serve, when braille is not an option (for whatever reason), audio has historically filled the gap. Audio renditions have become so common and powerful that development on more effective use of audio has continued. In recent years best practice in audio rendition has been codified in an ANSI/NISO standard, Z39.86. This same specification has been adopted internationally by a consortium of libraries for the blind and print handicapped called the *DAISY* Consortium (URL: www.daisy.org). In turn, this ANSI specification also serves as the basis for the U.S. legal mandate to provide accessible text books and curricular material in U.S. Schools, known as NIMAS (URL: nimas.cast.org).

Materials produced in audio include everything from novels for leisure reading, to newspapers, magazines, and technical reference material (in addition to curricular material). There are also national programs for creating and distributing such content across Europe, Canada, Australia, and Japan, as well as many other countries.

## 5.2 How ODF Fits In

ODF authoring applications are relevant to alternative media production, because alternative media production, including audio, almost always begin as does any

document, authored, spell-checked, proofread, and prepared for printing to paper. Because the ANSI/NISO and the NIMAS standards are XML specifications, there exists the opportunity to directly transform a properly prepared ODF document. Thus, ODF can readily serve as an authoring environment not only for print and e-documents, but for braille, large-print, and audio alternative renditions. ODF is, in fact, well suited to support commonly available authoring tools in the production of legally mandated alternative media.

### 5.2.1 Soft Page Breaks and Hard Page Numbering

[Ref to spec revision] at Section 4.2.6

Even the simplest content cannot be discussed in a group environment if there is no simple mechanism to point the group members to a particular location in the document. Pagination is the most common resolution to this problem, e.g. people will say something like: "please look at page 3 beginning at paragraph 2." However pagination needs specific coding in order to support alternative media where actual page numbers will be very different, e.g. print page 7 may be braille page 48.

Alternative media such as audio large print and braille require a mechanism that allows users to know, which page number is being referred to in the source document.

### 5.2.2 Structural Markup

Effective use of audio renditions requires that users have the ability to move quickly back and forth through the audio rendition based on the structure of the document. Traditional audio playback equipment provided fast forward and rewind mechanisms, but these are highly inefficient because time offsets are actually irrelevant to content.

Effective support of alternative rendering, and especially audio rendering, requires that the source document should be correctly tagged with structural markup. Both the ANSI and NIMAS specifications rely on XML markup to allow rendering agents to support quick movement forward and backward through content based on chapters, subsections, footnotes, paragraphs, and other structural elements.

The most effective devices allow users to adjust "levels" of navigation granularity, so that hierarchical structures such as X.Y.Z might be navigated at the X level, the Y level, or the Z level, at the user's option. This provides further emphasis on the importance of document structure when an ODF document is exported.

# 6 Glossary of Terms

*Accessibility API*

- An API (Application Programming Interface) designed to allow an assistive technology to interoperate with an application to provide alternative renderings, such as speech for people with disabilities.
- Examples: Microsoft Active Accessibility (MSAA), GNOME Accessibility API, Java Accessibility API, IAccessible2.

*Accessibility checker (or Accessibility evaluation tool)*

- A standalone software tool or office application function, which checks the accessibility of ODF files
- A user agent

*Assistive technology*

- Software applications or devices which assist people with disabilities who cannot access standard user interfaces. Provides enhancements or changed input and output methods, such as audio, Braille, software keyboard, magnification. Etc.

*ATK*

- ATK is an acronym for “Accessibility ToolKit”, an Accessibility API used to expose information on UNIX and GNOME desktops. ATK is implemented by GTK+, and exposed by XUL and UNO (and thus Firefox, Thunderbird, StarOffice, and OpenOffice.org) on UNIX desktops.

*AT-SPI*

- AT-SPI is an acronym for “Assistive Technology Service Provider Interface”. It is the platform and API-neutral inter-process communication mechanism and interface for assistive technologies on the UNIX desktop. It is used by UNIX assistive technologies in order to obtain accessibility information (and to control applications) that are in a different process address space than the assistive technology. Applications accessible via AT-SPI include GTK+ applications (implementing ATK), UNO applications (also implementing ATK), XUL applications (also implementing ATK), and Java/Swing applications.

*Braille display (or refreshable Braille display)*

- A device containing a row of Braille cells, and potentially additional Braille cells not in the row, and potentially a series of buttons, that together is used to represent text to someone who reads via touch – moving their finger(s) over the Braille cells. The Braille display is typically driven by a screen reader, which tells it what text to display on

the cells. Each Braille cell represents a character or contracted set of characters. Optional buttons on the devices may be used to communicate with the screen reader – allowing the user to scroll the text, or to “click” on text (typically when the buttons are positioned next to each Braille cell).

*Caption*

- A textual description for movies for the deaf or hard of hearing people. Open captions are always displayed on the screen, and closed captions can be hidden or displayed.
- A short textual description for non-textual objects (e.g. images and charts) or complicated visual structures (e.g. tables) on a screen. A caption is visible on the screen (in contrast to alternative text, which is not visible on the screen).

*Caret*

- A visible insertion point in a text editing interface. (See "insertion point")

*Compatibility (of User Interface)*

- The concept of user interface compatibility refers to a set of applications that can all be used with similar operations.
- For example, if the Control key plus the Right cursor key is assigned to move the caret to the end of the next word in several related programs, then this is part of their compatible user interface.

*Compatibility (of Accessibility API)*

- The concept of accessibility API compatibility refers to a set of applications that support the same accessibility API to control assistive technologies. Any combination of application and assistive technology that supports that API will work to allow users to access the compatible application.

*Converter*

- A user agent which converts documents into ODF files or ODF files into other types of documents.
- A tool to help authors or users of other applications.

*Cursor*

- A caret.
- A marker to indicate a position on a screen.
- Sometimes used as “mouse cursor”, in which case it is the marker to indicate the position of the mouse on the screen.

*Cursor keys*

- A set of directional keys used for directional operations (such as cursor movements or sub-menu operations).

*DAISY (Digital Accessible Information SYstem)*

- A standard format for digital talking books.
- URL: "DAISY Consortium" http://www.daisy.org/

*Dynamic text entry*

- A type of assistive technology for people with severe physical impairments that provides for extremely rapid (dynamic) entry of text for someone who can only move their head or eyes.
- Dasher, developed by the University of Cambridge Inference Group is the sole known example of this kind of assistive technology.

*FilterKeys*

- Sometimes though of as a built-in accessibility aid, other times as an assistive technology, it is a keyboard enhancement that optionally filters out repeated keystrokes if they occur within a short specified time. This is important for people with a variety of hand tremors who might accidentally press a key multiple times unintentionally. It is often used in concert with a repeat key setting. It was developed by the Trace Center, and is included with most desktop operating systems (including Windows, Macintosh, Solaris, Linux, and other UNIX variants).

*GNOME Accessibility API*

- An accessibility API designed for the GNOME desktop environment and UNIX desktops, derived from the Java Accessibility API. It was developed by the GNOME community under the leadership of Sun Microsystems, Inc.

*GTK+*

- GTK+ is an acronym for the “The GIMP ToolKit”, whilst “GIMP” is itself an acronym for the “GNU Image Manipulation Program”. GTK+ was originally developed as a graphical toolkit for the user interface elements of GIMP, but rapidly became an independent library used by GNOME desktop applications. GTK+ supports ATK.

*Java/Swing*

- The Java platform includes a variety of libraries, including the Swing user interface library for creating cross platform applications with rich graphical user interfaces. In the context of this document, “Java/Swing” means a Java application that utilizes the Swing user interface library. Java/Swing supports the Java Accessibility API.

*IAccessible2*

- An accessibility API derived from the GNOME and UNO Accessibility API; developed by IBM and submitted to the Free Standards Group for standardization.

- It complements MSAA by supporting additional functions necessary for the support of ODF applications, such as text controls, headings tables, hyperlinks, and relationships between objects.

*Insertion point*

- The position in a string of text where characters will inserted when a user inputs text by using text entry method. The insertion point should be marked with an input cursor or caret (for sighted users).

*Interoperability (of ODF editors)*

- The ability to exchange documents among a set of ODF editing programs. Any document generated by any member of a set of interoperable ODF editing programs can be read by any other ODF editing program in the set, and will be rendered as defined in the ODF specification.

*Java Accessibility API*

- An accessibility API for client-side Java applications (originally developed through a collaboration between IBM and Sun Microsystems, Inc.).
- It is implemented in GUI libraries written in Java, such as AWT and Swing.

*Landmarks*

- For non-visual navigation by using screen readers, a landmark refers to a virtual landmark. A blind user can memorize some part of a document to use as a landmark for navigation.
- Intentionally inserted supplemental landmarks can greatly help users to navigate in a document. The primary type of supplemental landmark in the ODF specification is the heading tag "Heading (<text:h>)".

*Microsoft Active Accessibility (MSAA)*

- A COM-based accessibility API for applications on Windows platforms developed by Microsoft Corporation.
- It is supported for Windows 98, Windows 2000, Windows Me, Windows XP, and Windows Vista.

*Mouth stick*

- A device, held in the mouth, that is used to press keys on a keyboard. This is typically used by someone who is unable to type with their hands.

*ODF editor*

- A user agent with editing capabilities for access and modification of ODF documents.
- An authoring tool.

*ODF generator*

- A user agent (software tool) on the client or server side that generates ODF files.
- An authoring tool.

*ODF reader*

- A user agent without editing capabilities, but enabling people to access ODF content.

*On-screen keyboard*

- A software program that presents a keyboard in a window on the screen, allowing the user to select keyboard keys under software control, as driven by some sort of input device (such as a switch device).
- The more sophisticated on-screen keyboard applications like the GNOME On-screen Keyboard (GOK) will also provide a variety of additional "keyboards": one showing the menu choices of the topmost application, one showing the toolbar items of the topmost application, one showing the user-interface elements (and visible hyperlinks) of the topmost application, etc.
- An assistive technology.

*Relationships*

- The concept of a relationship between visual objects on a screen, necessary to understand the screen contents.
- ODF has functionality to add relationships between objects.
- For Example. `form:for` (See 11.5.7), `draw:caption-id` (See 9.2.15)

*Synchronized media*

- A category of multimedia contents where various types of media content such as video, audio, text, and graphics are combined by using timing and synchronization controls.
- Examples: DAISY, SMIL .

*Screen magnifier*

- A software program used to magnify any object on screen (e.g. characters, images, etc.). Typical screen magnifiers not only magnify the contents of the screen, but also track the object the user is interacting with – the location of keyboard focus, the text caret, and the mouse cursor.
- An assistive technology.

*Screen reader*

- A program used to read aloud from office editor screens or office reader screens for non-visual users, as well as other applications.

Screen readers use text-to-speech to accomplish their reading. Often screen readers also support refreshable Braille displays.

- An assistive technology.

*ShowSounds*

- Sometimes though of as a built-in accessibility aid, other times as an assistive technology, it is a desktop setting that indicates to application software that it should convey audio information visually (e.g. by display text captions or informative icons). It was developed by the Trace Center.

*SlowKeys*

- Sometimes though of as a built-in accessibility aid, other times as an assistive technology, it is a keyboard enhancement that requires that a key be pressed for a specified period of time before the keystroke is accepted. This is important for people with a variety of hand tremors who might accidentally press a key briefly and unintentionally. It is often used in concert with a repeat key setting. Further, it is often used with ToggleKeys to emit a tone when a key is locked. Implementations of StickyKeys provide a variety of optional visual indications to show what key is made sticky. It was developed by the Trace Center, and is included with most desktop operating systems (including Windows, Macintosh, Solaris, Linux, and other UNIX variants).

*SoundSentry*

- Sometimes though of as a built-in accessibility aid, other times as an assistive technology, it is a desktop enhancement that causes a visual indication to be made whenever sound is used to indicate a warning or error (as opposed to playing general WAV files). This is important for people who are deaf. Common visual indication options are to flash the entire screen, flash the window, flash the window titlebar, and flash the desktop-wide menubar. It was developed by the Trace Center, and is included with most desktop operating systems (including Windows, Macintosh, Solaris, Linux, and other UNIX variants).

*Speech recognition*

- Software that is capable of recognizing human speech. Speech recognition is typically used as part of an application that provides speech control of other applications, and/or dictation.

*StickyKeys*

- Sometimes though of as a built-in accessibility aid, other times as an assistive technology, it is a keyboard enhancement that will lock (or make 'sticky') the various modifier keys on a keyboard, such as SHIFT, CONTROL, and ALT. This is important for people who can only press one key at a time – such as users of a mouth stick. It is often used in concert with a repeat key setting. It was developed by the Trace

Center, and is included with most desktop operating systems (including Windows, Macintosh, Solaris, Linux, and other UNIX variants).

*Switch device*

- A hardware switch designed for someone with limited mobility. Switch devices are commonly mounted on wheelchairs, where they are activated by a user shrugging their shoulder, moving their head, or otherwise moving some part of their body against the switch. There are also sip-and-puff switches that are activated by sipping and/or puffing through a straw. Switches are commonly used with on-screen keyboard software in order to enter text and control other applications on a desktop.

*ToggleKeys*

- Sometimes though of as a built-in accessibility aid, other times as an assistive technology, it is a keyboard enhancement that requires that a key be pressed for a specified period of time before the keystroke is accepted. This is important for people with a variety of hand tremors who might accidentally press a key briefly and unintentionally. It is often used in concert with a repeat key setting. Further, it is often used with ToggleKeys to emit a tone when a key is locked. Implementations of StickyKeys provide a variety of optional visual indications to show what key is made sticky. It was developed by the Trace Center, and is included with most desktop operating systems (including Windows, Macintosh, Solaris, Linux, and other UNIX variants).

*TTS (Text-to-Speech)*

- A speech synthesis system: often called TTS because of its ability to convert text to speech.

*UI Automation*

- An alternative accessibility API to MSAA developed by Microsoft Corporation.
- It is available on all systems that support the Windows Presentation Foundation (WPF).
- It provides functions for creating automated testing tools, and assistive technologies.

*Universal Access API*

- An accessibility API developed by Apple Inc.
- It is included within the “Universal Access” features in Mac OS X 10.4 or later.

*UNO Accessibility API*

- An accessibility API developed for OpenOffice.org applications, derived from the Java Accessibility API. It was developed by the

OpenOffice.org community under the leadership of Sun Microsystems, Inc.

- It is translated into the Java Accessibility API on Windows, and into the GNOME Accessibility API on UNIX systems systems.

*UNO*

- UNO is an abbreviation for “Universal Network Objects”, a user interface library and toolkit developed by the OpenOffice.org project for use in the OpenOffice.org and StarOffice applications. UNO supports the UNO Accessibility API.

*User agent (or ODF user agent)*

- Any type of software tool which reads or writes ODF files.

*Voice office editor*

- A user agent with editing capabilities for users who prefer audio interactivity.
- An office editor.
- An assistive technology.
- An authoring tool.

*Voice office reader*

- A user agent without editing capabilities for users who prefer audio interactivity.
- An office reader.
- An assistive technology

*.XUL*

- XUL is an abbreviation for “XML User Interface Language”, a user interface markup language developed by the Mozilla Foundation for use in Mozilla family software applications (including the Firefox web browser and the Thunderbird e-mail application). XUL-runtimes (Firefox, XUL Runner, etc.) support the ATK accessibility toolkit on GNOME systems, and the MSAA and IAccessible2 interfaces on Windows. Work is underway to support the Apple Accessibility API as well.

# Appendix A. Acknowledgments

**Contributors:**

Chieko Asakawa, IBM
Pete Brunet, IBM
Hiro Takagi, IBM
Richard Schwerdtfeger, IBM
David Clark, Individual
Stephen Noble, Individual
David Pawson, Individual
Janina Sajka, Individual
Peter Korn, Sun Microsystems, Inc.
Malte Timmermann, Sun Microsystems, Inc.