Open Document Format for Office Applications (OpenDocument)v1.2

Part 2: Recalculated Formula (OpenFormula) Format – Annotated Version

Pre-Draft12, 8 May 2009

Specification URIs:

This Version:
OpenFormula-v1.2-draft12.odt at http://www.oasis-open.org/committees/office

Previous Version:
-

Latest Version:
http://www.oasis-open.org/committees/office

Latest Approved Version:
-

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

Chairs:
David A. Wheeler (Formula Sub Committee Chair),
Robert Weir, IBM
Michael Brauer, Sun Microsystems, Inc.

Editors:
David A. Wheeler,
Eike Rathke, Sun Microsystems, Inc.
Robert Weir, IBM

Related Work:
-

Declared XML Namespaces:
A list of XML namespaces declared by this specification is available in section 1.3.

Abstract:

Copyright © OASIS Open 2006. All Rights Reserved.

This is the specification of the Open Document Format for Office Applications (OpenDocument) format, an open, XML-based file format for office applications, based on OpenOffice.org XML [OOo].

This specification has three parts.

Part 1 defines an XML schema for office applications and its semantics. The schema is suitable for office documents, including text documents, spreadsheets, charts and graphical documents like drawings or presentations, but is not restricted to these kinds of documents.

Part 2 (this document) defines a formula language to be used in OpenDocument documents.

Part 3 defines a package format to be used for OpenDocument documents.

Status:

This document is a preliminary draft. Its outline is incomplete and does not provide an outlook to the topics addressed by OpenDocument v1.2.

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 current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

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

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

(www.oasis-open.org/committees/office/ipr.php.

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

Note: This is the *annotated* version of this specification, which includes *non-normative* information (notes, rationales, and TODO/TBD). These annotations are not included in the final official specification, and thus are not normative, but they are available to implementors as guidance.

To show only the normative specification: Modify the text at the end of the first line of this document (after “OASIS”), which has a text setting for the value “Notes”. If “Notes” is set to 0 (off), the annotations (including the one your are viewing now) will not be displayed.

Copyright © OASIS Open 2006. All Rights Reserved.

Notices

Copyright © OASIS® 2002–2007. All Rights Reserved. OASIS trademark, IPR and other policies apply.

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”, “Open Document Format” and “ODF” 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.

Copyright © OASIS Open 2006. All Rights Reserved.

Table of Contents

Copyright © OASIS Open 2006. All Rights Reserved.

Copyright © OASIS Open 2006. All Rights Reserved.

Copyright © OASIS Open 2006. All Rights Reserved.

1 Introduction

1.1 Introduction

This is the specification for OpenFormula, an open format for exchanging recalculated formulas between office applications, particularly formulas in spreadsheet documents. OpenFormula defines the types, syntax, and semantics for recalculated formulas, including many predefined functions and operations.

The purpose of this specification is ensure that people *can own their data*, instead of having their data controlled by a vendor who controls the data format. Using OpenFormula allows document creators to change the office application they use, exchange formulas with others (who may use a different application), and access formulas far in the future, with confidence that the recalculated formulas in their documents will produce equivalent results if given equivalent inputs.

Other advantages of this specification are:

1. Users can be confident in its credibility, because it was developed through the cooperation of many different spreadsheet implementors and users in a completely open manner. Development was done though public media (particularly publicly-readable mailing lists), and drafts were constantly available for public review and critique. No one vendor dominated its development.

2. There is no limitation on who may implement this specification. In particular, there is no question that both closed and open source software, using their typical licenses, can implement OpenFormula.

3. It is designed to allow any developer to innovate, without interfering with other applications. This is particularly obvious in the function naming scheme for application-specific functions. This enables different application developers to innovate by creating new functions without interfering with other applications or with standard functions. Other application implementors can implement a function that first appeared elsewhere, using the prefixed name of the original function's creator. This enables a standard for the function to emerge later, which may have modified semantics based on the experience of the other applications.

4. It is useful to a very broad range of developers, from relatively small applications (typical for resource-constrained systems) through large applications with a large number of functions. This is accomplished through the careful predefinition of different conformance groups, particularly Small, Medium, and Large, based on analysis of many different real-world spreadsheet applications.

5. It provides an unusually rich set of capabilities. This is because this specification was derived through the examination of a large number of different spreadsheet applications, instead of including only the function definitions from a single application. It includes an unusually rich set of function definitions, including functions such as ARABIC, BASE, BITAND, COT, COTH, GAMMA, IFERROR, and XOR. It also defines how to provide references to unnamed cells in external spreadsheets, enabling reuse of external spreadsheets when they do not define relevant named expressions.

6. Its function definitions have been developed by analyzing both documentation and test cases of a variety of applications on various platforms.

7. Implementation-defined areas were specified based on an analysis of the differences between real applications, instead of over-constraining the specification (which could imply a requirement to use only one specific product). Some applications treat logical

Copyright © OASIS Open 2006. All Rights Reserved.

values TRUE() and FALSE() as a distinct type from numbers, while others treat them as 1 and 0 respectively. Similarly, some automatically convert text to numbers in a number context, while others do not (for a variety of reasons, e.g., automatic conversions often produce incorrect answers when shared internationally due to differences in the decimal point character). Different applications produce different error values in the same circumstance, and this specification explicitly permits certain kinds of variance (while making clear to document developers that there is such variance, and how to deal with it).

8. It is intentionally designed to work well into the future, instead of only being able to capture the past. For example, the format allows an arbitrary number of columns, as opposed to imposing an implied limit from its cell addressing syntax. It is also able to specify unusual range names.

9. It is international. It supports arbitrary names of named expressions and sheets, including international characters - so users can use names that are natural for them, no matter what their language. It also does not presume that strings containing numbers use "." as the decimal point, since this is not true in many locales.

OpenFormula is intended to be a supporting document to the Open Document Format for Office Applications (OpenDocument) format, particularly for defining its attributes `table:formula` and `text:formula`. It can also be used in other circumstances where a simple, easy-to-read infix text notation is desired for exchanging recalculated formulas.

OpenFormula does ***not*** define:

- the user interface. User interfaces may use different syntaxes, different function names, and/or different parameter orders. In particular, OpenFormula uses ";" as a parameter separator (which works well internationally), while some user interfaces use "," instead. OpenFormula surrounds all cell references with square brackets to eliminate ambiguity and to allow very wide tables; user interfaces often do not display them. Many applications translate the displayed function names (e.g., to the local language based on the system's locale). OpenFormula uses the symbols "~" for reference union, "!" for reference difference, and "." to separate sheetnames from cell addresses; some user interfaces may differ. However, OpenFormula's syntax is very similar to typical user interfaces, reducing the risk of misunderstanding
- internal representations or implementations
- a general notation for mathematical expressions, including mechanisms to control the display format. This specification has a fundamentally different role from mathematical display notations such as MathML
- a full-fledged programming language. OpenFormula formulas calculate a result and return it. By design, most operations and functions are free of side-effects, and it is not possible to "loop forever" inside a single formula.
- a special "summarize data" mechanism that presents many different statistical values of a set of data (e.g., average, minimum, first quartile, median, third quartile, maximum, standard deviation, and mode). If desired, applications **may** implement such a mechanism using a "wizard" to create a set of (labeled) formulas for each of these statistical values, using the formula language defined here. Such summaries can then be exchanged normally.
- a mechanism to detect likely errors in formula. However, by defining the format and semantics of formulas, it enables the creation of them.

Note: This specification is derived from actual practice in industry. It was especially influenced by the OpenOffice.org exchange syntax and by the semantics of Microsoft Excel, but many other spreadsheet implementations were considered including (alphabetically) Corel's WordPerfect suite, Document To Go's Sheet to Go, GNOME Gnumeric, IBM/Lotus 1-2-3, KOffice's Kspread,

Copyright © OASIS Open 2006. All Rights Reserved.

and WikiCalc. This was done to simplify transition of spreadsheet documents to this format. Also, Winer's Law of the Internet (by Dave Winer) claims that "Productive open work will only result in standards as long as the parties involved strive to follow prior art in every way possible. Gratuitous innovation is when the standardization process ends, and usually that happens quickly."

Implementations use OpenFormula as a method to exchange recalculated formulas. Once an implementation reads a formula in this format, it may choose to represent the formula internally in an arbitrary way (such as a bytecode sequence, compiled machine code, or a tree of nodes). Formulas are executed whenever they are needed to compute a specific result.

OpenFormula is not a full-fledged programming language. It is specifically designed to describe how to calculate a single result, which it returns as its answer. OpenFormula is, in general, side-effect-free; it does not have built-in operations to "assign" a value, and unless otherwise noted its functions do not have side-effects. Built-in functions, when passed the same values, generally produce the same result (with the notable exception of random number functions like RAND()). It is designed so that it is not possible to describe a calculation in a single function that can "loop forever." By design, all formula calculations that only reference built-in functions must eventually end (though this may take a long time for a few computationally-intense functions, or end in failure if they involve references to external values that cannot be acquired). In a spreadsheet, endless loops can be created indirectly through multiple formulas, because two or more formulas may (through references) create a circular dependency. Implementations typically handle such cycles specially, either (1) forbidding them or (2) requiring a special setting before allowing them which also prevents them from becoming endless. For example, implementations of OpenDocument normally forbid circular references unless the `<table:iteration>` element is set, and if set, iterations are allowed up to a specified maximum number of iterations.

OpenFormula's format is designed to be useful as an attribute value within an XML document. It is *not,* itself, in XML, but instead uses the traditional mathematical infix notation. Formulas used for recalculation have not traditionally been written using XML. Instead, computer-recalculated formulas are traditionally written using a modified mathematical infix notation such as "=value*5+2". This non-XML format is much easier for humans to understand (because it is the form people have used for decades), is shorter than XML formats, and there are many tools specifically designed to handle these formats. This is different from specifications such as MathML, which are designed to support display and not necessarily recalculation. Formulas not used for (re)calculation are not the primary subject of this specification.

This specification first describes conformance issues and and the basic formula processing model. It then presents the types that values may have (as defined by this specification) and the expression syntax. This is followed by the bulk of this document, defining the implicit conversion operators, standard operators, and functions.

Note: Spreadsheets can contain errors. If spreadsheets are used for extremely critical decisions, it is wise to use tools that try to detect such errors (some are built into some spreadsheet implementations; other tools are external). More information about errors in spreadsheets can be found in locations such as the article "The Risky Business of Spreadsheet Errors" by Ivars Peterson, the European Spreadsheet Risks Interest Group, and Ray Panko's Spreadsheet Research (SSR) page. This specification does not directly implement a defect detection tool, but it is key to enabling and improving such tools. By publicly defining a common format for spreadsheet formulas, and by publicly defining their semantics, this specification makes it much easier to build defect detection tools for spreadsheets. The specification even points out potential issues (e.g., precedence of ^ vs. unary -), making it easier to build tools. Thus, this specification can be viewed as a key step in countering defects in spreadsheets.

1.2 Document Notation and Conventions

Within this specification, the key words "**shall**" and "**shall not**" (for requirements), "**should**" and "**should not**" (for recommendations), "**may**" and “**need not**” (for permissions), and “**can**” and

Copyright © OASIS Open 2006. All Rights Reserved.

"**cannot**" (for statements of possibility or capability) are to be interpreted as described in Annex H of [ISO/IEC Directives] (part 2).

Note: Information within a *Rationale:*, *Note:*, or *TODO/TBD:* section is not normative:

- Rationale explains why a part of the specification is defined that way.
- Notes provide other information, such as tips for implementation or information on what various existing implementations do.
- TODO/TBD identifies something left to do, possibly including the pros and cons for a decision yet to be made.

It is expected that there will be an unannotated version of this document, without this information, which will be the version proposed for standardization, as well an annotated version *with* this information (but not formally standardized). It is expected that the official printed form will *not* include this information, but that the annotations will be included as hidden text.

This information is embedded throughout the document to prevent questions from being asked repeatedly during specification development, and to aid implementors in avoiding common mistakes. Much discussion about existing implementations is included; where practical, existing implementations should help guide any standard, and these additional notes should help ensure that this is true.

In English the plural of formula can be "formulas" or "formulae"; this document uses the more common plural in English, "formulas".

Rationale: Both the New York Times "Everyday Dictionary" (1982) and Merriam-Webster's "Webster's Ninth New Collegiate Dictionary" (1983) declare that both "formulas" and "formulae" are acceptable. A Google search on 2005-12-30 found 24,800,000 pages with the term "formulas" as opposed to 6,810,000 for "formulae", so we chose to use the more common spelling. Please change all uses of the term "formulas" to "formulae" when translating this specification into Latin.

1.3 Namespace

This specification defines a particular formula syntax that is often contained in XML attributes. In this case, the attribute value will typically begin with "="; normally the namespace will not be referred to. However, it is legal to specifically identify a namespace, and to include a namespace prefix in the attribute. If this occurs, the "of:" is typically used, and the following namespace is used:

```
urn:oasis:names:tc:opendocument:xmlns:openformula:1.0
```

For more information about XML namespaces, please refer to the *Namespaces in XML* specification [xml-names]. Implementations **may** also accept formula syntaxes other than OpenFormula, and they **may** accept various compatible extensions to the default OpenFormula syntax.

Rationale: This provides additional flexibility so that other languages can be used when desired. The selector is not considered part of OpenFormula itself, so that specifications that use OpenFormula can choose how to best include this selector. In OpenDocument, formulas beginning with "=" are in OpenFormula format; other formats must use a namespace identifier.

Note that the namespace selector is in the attribute *value*, and not used as a prefix for an entity or attribute name. This is intentional, because in all cases, these are formulas; the prefix simply gives more information on which formula language is being used. While less common, this way of using selectors is done elsewhere for similar issues. For example, the XSLT specification uses QNAMEs for multiple purposes (http://www.w3.org/TR/1999/REC-xslt-19991116#qname), for example, QNAMEs are used in XSLT for extension functions (http://www.w3.org/TR/1999/REC-

Copyright © OASIS Open 2006. All Rights Reserved.

xslt-19991116#section-Extension-Functions). Another example is the W3C "CURIE Syntax 1.0" draft specification (http://www.w3.org/TR/2008/WD-curie-20080506).

Note: For OpenDocument files written by OpenOffice.org versions prior to this specification (including versions 2.0, 2.0.1, 2.0.2, and 2.0.3), the typical prefix and namespace used for spreadsheet formulas is xmlns:oooc="http://openoffice.org/2004/calc". Early test sheets used this namespace instead, since applications were not aware of the OpenFormula namespace.

Rationale: Originally, there was a section 1.4 titled “Normative References”. However, none are noted at this time, so the section heading (for an empty section) has been removed. Earlier, we considered adding a reference to some of the various national and international floating-point specifications, but eventually no need to do so was identified; in particular, this specification does not actually *require* implementations of those specifications, though implementations which do so can certainly implement this specification.

There is no need to reference the OpenDocument specification, since this document is a volume in the OpenDocument specification. However, the OpenFormula specification is specifically written so that it can be referenced and used outside of OpenDocument, should someone choose to do so, to encourage broad use and reuse.

Copyright © OASIS Open 2006. All Rights Reserved.

2 Conformance

Applications **may** implement subsets or supersets of this OpenFormula specification. An application **shall** only claim to conform to a given function, operator, or group if the application completely meets *all* of its requirements as defined in this specification. Applications **may** (and typically do) implement additional functions beyond those defined in this specification. Applications **may** support additional formula syntax, additional operations, additional optional parameters for functions, or make certain function parameters optional when they are required by this specification. Applications **should** clearly document their extensions in their user documentation, both online and paper, in a manner so users would be likely to be aware when they are using a non-standard extension.

This specification's text is written as a description of the requirements of an implementing application. However, documents (data files) containing formulas can also comply or fail to comply with this specification. Documents with OpenFormula formulas **may** use subsets or supersets of OpenFormula. A document **may** reference a nonstandard function by name, or depend on implementation-defined behavior, or on semantics not guaranteed by this specification. Thus, this specification discusses what is required for a document to assert that it is a *portable document.* A portable document **shall** only depend on the capabilities defined in this specification, and **shall not** depend on undefined or implementation-defined behavior. A portable document **shall** only claim to conform to a given group if the document only depends on the capabilities of the given group.

This specification defines OpenFormula formulas in terms of a canonical text representation used for exchange. OpenFormula formulas are normally defined as attributes in XML documents. When OpenFormula formulas are attributes in XML documents, characters **shall** be escaped as required by the XML specification (e.g., the character & shall be escaped in XML attributes using notations such as &). In OpenDocument, OpenFormula formulas are stored in XML attributes, particularly `table:formula` and `text:formula`; these XML documents are typically compressed using the zip format, as further described in the rest of the OpenDocument specification. These escape and compression mechanisms are outside the scope of OpenFormula. All string and character literals references by this specification are in the value space defined by [ISO10646]; thus, “A” is U+0041, “Z” is U+005A, and the range of characters “A-Z” is the range U+0041 through U+005A inclusive.

2.1 Predefined Groups

To encourage interoperability, groups of functionality have been predefined. These groups make it easier to determine if a given application will be able to correctly process a given document. Acquirers **can** select applications depending on whether or not the application conforms to the group(s) necessary for the acquirer's circumstances. Document developers **can** choose to only use capabilities defined in certain groups when creating their documents.

Functions are grouped by purpose in this specification. Any application that implements all of the functions listed in that group, as defined in this specification, conforms to that group. These are simply the second-level headings of section 6.3 and later, e.g., there is an array/matrix functions group, database functions group, financial functions group, and so on.

There are also several predefined groups that include a number of common functions, where each group is a strict superset of previous groups:

- Small: This group includes approximately 100 functions, and includes the basic functions that are very widely implemented in spreadsheet applications, even in resource-constrained

Copyright © OASIS Open 2006. All Rights Reserved.

environments. It does not require support for inline arrays, complex numbers, nor the reference union operator.

- Medium: This group provides approximately 200 functions; many current desktop implementations meet or nearly meet the requirements of this group. It does not require support for inline arrays or complex numbers, but it does add the reference union operator.
- Large: This group provides over 300 functions, and requires support for inline arrays, complex numbers, and the reference union operator.

Note that some applications **may** be able to read and write a document that states that it requires a larger group, because applications may partially implement a larger group.

Note: Future versions of this specification may add a “huge” group (over 400 functions) and/or a “tiny” group (e.g., one without database functions and so on).

Rationale: Here are some of the drivers for the various groups:

- Small is driven by PDAs, cell phones, etc. It represents the capabilities and functions implemented by such diverse applications as Sheet To Go (which runs on PalmOS PDAs), OpenOffice.org 2 Calc, and Microsoft Excel 2003. In general, if any implementation did not implement a particular function or semantic, that requirement was moved outside of Small (with a few exceptions, such as requiring both ISERR and ISERROR).
- Medium is a "majority intersection" of the capabilities in common desktop spreadsheet programs (e.g., *most* spreadsheet implementations for the desktop provide those capabilities), considering implementations such as Microsoft Excel 2003, Corel Word Perfect Office 12, IBM/Lotus 1-2-3 version 9.8, OpenOffice.org Calc, and Gnumeric.
- Large is driven by what is needed to be compatible with market-leading spreadsheet applications, such as Microsoft Excel 2003 and OpenOffice.org.
- Huge is almost the union of capabilities of common desktop spreadsheet programs; it is not currently defined in this specification, but a future version might add it.

Having groups for compliance lets applications quickly achieve some level of compliance that is sufficient for many user needs, yet be able to distinguish themselves if they provide an especially rich set of functionality. In particular, it is expected that many applications will meet the small group almost out of the box (once they support the file exchange and syntax). The Medium group gives implementations a relatively easy goal (if they don't already implement it). The Huge group is a steep goal, but even if spreadsheet implementations that do not implement everything in Huge can implement a subset with the same semantics -- and thus be able to exchange that information with other programs that implement the same subset. Additional groups “Tiny” and “Colossal” could be added on both ends of the grouping, if appropriate.

Here are some of the known weaknesses of current spreadsheet implementations as compared to Large:

- KSpread's MOD function handles negative numbers differently than other spreadsheets, and KSpread 1.4.2 incorrectly computes VALUE (e.g., VALUE("6") is 0 instead of 6). This is old information and both issues have probably already been fixed.
- OpenOffice.org 2.0's LOG function requires the second parameter; here it is optional. This is already fixed as of OpenOffice.org 2.4.

For more about the differences between Excel and Lotus 1-2-3, see “Calculation differences between Microsoft Excel and Lotus 1-2-3 formulas”. That document has many errors in it, however: Excel does not ALWAYS treat text as an error in a Number context, in fact, if given A4+A5, and A4 is the text "100", Excel 2002 will silently convert the text to a number. The note does report that "Excel 2000 and later versions contain functions for compatibility with Lotus 1-2-3

Copyright © OASIS Open 2006. All Rights Reserved.

Release 4.0 and later. The "A" functions— AVERAGEA, MAXA, MINA, STDEVA, STDEVPA, VARA, and VARPA— calculate results by using all of the cells in a range (including blank cells), cells that contain text, and cells that contain the logical values TRUE or FALSE."

Complex numbers are placed in the “large” group, because not all applications support them, and even those that do sometimes are require additional steps. Excel 2003 can handle them, but only when the Analysis Toolpak is installed (by going to Tools/Add-ins). Gnumeric can handle them. There is an extra package for OpenOffice.org 2 for handling complex numbers. Unfortunately, the approach to Complex numbers used by Excel and others, using text values, is incredibly convoluted. Instead of creating a new type (Complex) and allowing functions to handle the new type, such applications require that you use a whole new set of functions, even for operators that are normally expressed with infix notation. Many spreadsheets cannot use infix operators like "*" to multiple complex numbers. Thus, to compute e^PI*i, instead of saying EXP(PI*i), you have to say IMEXP(IMPRODUCT(PI(),COMPLEX(0,1))). Not all functions have an IMxyz() equivalent, making complex number use even more complicated. However, that really is what applications require. This specification is written so that portable approaches work, but is carefully written to permit future applications to add a complex number (as a distinct type, type of number, or subtype of number), and permit operators like “+” to work on them directly.

The following precisely defines the three predefined groups Small, Medium, and Large.

2.1.1 Small Group

For an application to claim that it conforms to the “Small” group of functionality, it **shall**:

- Support at least the limits defined in the “Basic Limits” section.
- Support the relevant syntax required in these sections on syntax: Criteria; Namespace Selection; Basic Expressions; Constant Numbers; Constant Strings; Operators; Functions and Function Parameters; Nonstandard Function Names; References; Simple Named Expressions; External Named Expressions; Errors; Whitespace
- Implement all implicit conversions for its applicable types: at least Text, Conversion to Number, Reference, Conversion to Logical, and Error
- Implement the following operators (which are all the operators except reference union (~)): Infix Operator Ordered Comparison ("<", "<=", ">", ">="); Infix Operator "&”; Infix Operator "+”; Infix Operator "-”; Infix Operator "*”; Infix Operator "/”; Infix Operator "^”; Infix Operator "=”; Infix Operator "<>”; Postfix Operator “%”; Prefix Operator “+”; Prefix Operator “-”; Infix Operator Reference Intersection ("!"); Infix Operator Range (":")
- Implement at least the following functions as defined in this specification: ABS; ACOS; AND; ASIN; ATAN; ATAN2; AVERAGE; CHOOSE; COLUMNS; COS; COUNT; COUNTA; COUNTBLANK; COUNTIF; DATE; DAVERAGE; DAY; DCOUNT; DCOUNTA; DDB; DEGREES; DGET; DMAX; DMIN; DPRODUCT; DSTDEV; DSTDEVP; DSUM; DVAR; DVARP; EVEN; EXACT; EXP; FACT; FALSE; FIND; FV; HLOOKUP; HOUR; IF; INDEX; INT; IRR; ISBLANK; ISERR; ISERROR; ISLOGICAL; ISNA; ISNONTEXT; ISNUMBER; ISTEXT; LEFT; LEN; LN; LOG; LOG10; LOWER; MATCH; MAX; MID; MIN; MINUTE; MOD; MONTH; N; NA; NOT; NOW; NPER; NPV; ODD; OR; PI; PMT; POWER; PRODUCT; PROPER; PV; RADIANS; RATE; REPLACE; REPT; RIGHT; ROUND; ROWS; SECOND; SIN; SLN; SQRT; STDEV; STDEVP; SUBSTITUTE; SUM; SUMIF; SYD; T; TAN; TIME; TODAY; TRIM; TRUE; TRUNC; UPPER; VALUE; VAR; VARP; VLOOKUP; WEEKDAY; YEAR

Rationale: The list of functions in the “small” group is based on Richard Kernick's email of 2005-10-20 on the OpenFormula mailing list, identifying the list of functions supported by Documents to Go for the Palm, version 6. He proposed that this list would be an excellent subset, since it demonstrated (through an actual implementation) what functions would best support a significant number of spreadsheet users in small environments. In general, we want to emphasize what

Copyright © OASIS Open 2006. All Rights Reserved.

industry is doing, instead of pretending, so unless suggested otherwise we used actual implementations as our guide.

Changes were then made based on examination of various implementations. ERROR.TYPE is not universally portable (e.g., it's named ERRORTYPE in OOo2 with completely different results), so it was moved out of this group. Instead of CONCATENATE people normally use &, so it was removed from this group as well.

Rationale: A few of these functions are not in Quattro Pro or Lotus 1-2-3.In particular, there is an ISERR/ISERROR issue. Both Quattro Pro and Lotus 1-2-3v9 have a function named ISERR; neither has a function named ISERROR. Even more confusingly, in Lotus 1-2-3v9, ISERR has the functionality of ISERROR instead of ISERR. This is because in Lotus 1-2-3v9, @ISERR(@NA) is 1 (True). In Quattro Pro, Excel, OpenOffice.org, SheetToGo, etc., ISERR(NA()) is 0 (False); this is written as @ISERR(@NA) in Quattro Pro.The function ISERROR could be moved to a higher level, leaving only ISERR at level 1; but this would not handle Lotus 1-2-3v9.8, since it would map its ISERR to this specification's ISERROR. Removing ISERROR would require rewriting of many level 1 tests as well as the testsuite generator. In some sense this means that "NA" isn't considered an "error" by systems that only have ISERR, since the only True/False detection system skips NA. But by defining ISERR with the semantics as has been done, the *model* defined in this specification is still correct. Conversely, ISERR could be moved to a higher level, which would mean that Quattro Pro does not meet level 1. There is no perfect solution; it is impractical to have a group lacking both ISERR and ISERROR. Since both functions are quite valuable, and both are trivial to implement, both are required for even the small group.

It is unfortunate that the functions ISERR and ISERROR have such similar names, but this really is standard practice, so we've simply documented it that way.

Note: Other functions not in Quattro Pro are: COLUMNS, COUNTA, DCOUNTA, DSTDEV, LOG10, PRODUCT. In some cases the "nonmatches" are simply different names. E.G., Quattro Pro's ISSTRING maps to ISTEXT, LENGTH to LEN, REPEAT to REPT, and so on.

Lotus 1-2-3v9.8.1 represents AND, OR, and NOT as operators but has them. It has no distinguishable POWER function, and must use the infix operator. Functions not in Lotus 1-2-3v9.8.1 are: COLUMNS, COUNTA, DCOUNTA, DEGREES, DPRODUCT, ISLOGICAL, RADIANS, SUBSTITUTE. Many functions are just renamed, e.g., ISEMPTY becomes ISBLANK.

Note: John Cowan reported on 2005-10-19 that he had examined the following implementations for their time/date functions:

Microsoft Excel 2003	internal help
Gnumeric	list on web site
KSpread	posted to this list
Quantrix Modeler 2.0	manual on web site
OpenOffice.org 2.0	manual draft on web site

Cowan reported that "Quantrix Modeler is a spreadsheet-like application that descends conceptually from Lotus Improv, so it is an offshoot of the main line of tradition and therefore valuable as a cross-check."

All 5 of them implemented the following, and therefore were recommended for level 1:

DATE, DATEVALUE, DAY, DAYS360, HOUR, MINUTE, MONTH, NOW,

SECOND, TIME, TIMEVALUE, TODAY, WEEKDAY, YEAR.

He proposed the following functions for level 2. They are not present in Quantrix, and are available in OpenOffice.org only when the Analysis AddIn is present:

Copyright © OASIS Open 2006. All Rights Reserved.

EDATE, EOMONTH, NETWORKDAYS, WORKDAY, YEARFRAC.

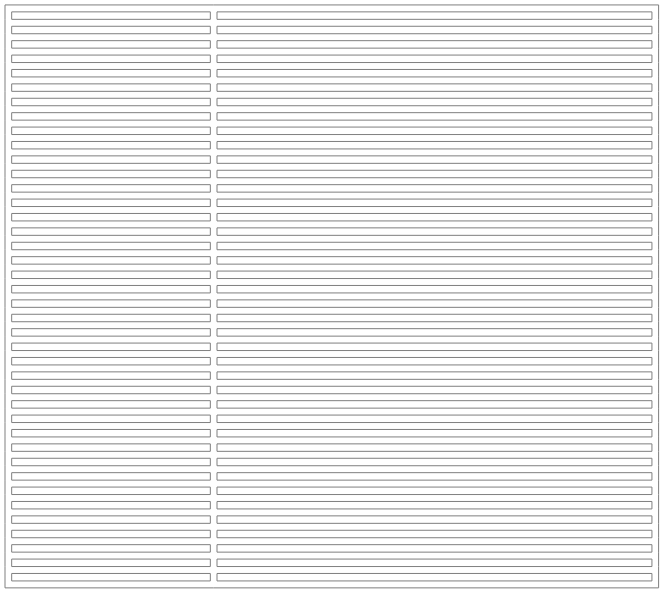
Applications conforming to the small group **may** limit their references to contain at most one area, and **may** choose to not implement other capabilities such as inline arrays, complex numbers, and the reference union operator.

If the formula is embedded in an OpenDocument document, applications conforming to the small group **shall** implement the following calculation settings (that is, each setting **shall** have the effect as defined in this specification):

1. `table:case-sensitive`
2. `table:precision-as-shown`
3. `table:search-criteria-must-apply-to-whole-cell`
4. `table:automatic-find-labels`
5. `table:use-regular-expressions`
6. `table:use-wildcards`
7. `table:null-year`
8. `table:null-date`

Note that even this group **shall** support international characters for named expression identifiers, since it is important that this specification be useful worldwide. However, this specification does not mandate a user interface, so a resource-constrained application **may** choose to not show the traditional glyph (e.g., it may show the ISO 10646 numeric code instead).

Note: The `table:use-wildcards` attribute was proposed to the ODF-TC on 2007-03-08 by Eike Rathke, and was eventually accepted. It was confirmed to be documented in OpenDocument 1.2 draft dated 28 April 2008.



2.2 Variances (Implementation-defined, Unspecified, and Behavioral Changes)

Copyright © OASIS Open 2006. All Rights Reserved.

2.3 Test Cases

To reduce the risk of misunderstanding a requirement, and to increase the likelihood of correct implementation, this specification includes a large number of test cases that are *normative* (that is, they are part of the specification). In particular, every function and operator has *at least* one test case, and typically many test cases. An implementation **shall** pass all of the test cases for a given function or operator (unless otherwise *specifically* noted in the text) to be able to claim conformance with that function or operator. No set of test cases can be complete, so to claim conformance for a given function or operator, implementations **shall** meet all the requirements of this specification of it, even if there is no specific test case for some aspect of it.

For the test cases, all calculation settings are at their defaults, *except* that case-sensitive is false (so "Hi"="HI" is true) and search-criteria-must-apply-to-whole-cell is false.

A very few test cases depend on a specific locale. In these cases, the presumed locale is "en_US" (American English). This does not imply that any locale is “better” than another; this is simply done to reduce the costs of checking compliance, and to be sure that at least one locale works correctly. Implementations **should** support a large range of locales. In general, test cases have been defined to be locale-independent where practical to do so.

Test cases are defined by a table. For each test case, the following are defined:

- Expression: The expression being computed for that test case, in OpenFormula format. This begins with an = sign for non-array formulas. Array formulas begin with {=, and end with a matching } (note that expressions are not actually stored this way in OpenDocument). In most cases, an effort has been made to write expressions so that the results have a finite representation in both base 2 and base 10 (e.g., integers or fractions of powers of 2) to simplify comparisons (with the exception of test cases specifically to test representations). Many test expressions involving trigonometric functions involve PI() to make the results easier to compare. All quotation marks are straight (so if an expression appears to have curling quotes, it really contains straight quotes). The expression does *not* display any necessary XML encoding when it is stored in an XML document; thus an expression will show "<", not "<".
- Result: The required result of the expression in a compliant implementation of OpenFormula. This is one of the following patterns (ignoring leading and trailing whitespace):
- *True* or *False*: The calculated expression **shall** be equal to the value of TRUE() or FALSE(), respectively, as determined using the “=” operator. Note that in applications with a distinct logical type, 1 is not equal to TRUE().
- "text": The calculated expression **shall** be exactly equal (including case) to the text, as determined by the EXACT function. There is no difference in semantics between curling quotes and straight quotes that surround the text.
- Number: A number, which begins with a digit (0-9) or a minus sign, in OpenFormula syntax (so the period is the decimal point, and scientific notation is allowed). If there is no “±” symbol in the result, the calculated expression **shall** be equal to the number as determined using the “=” operator. The number may end with “±” followed by an error range, which is either a second number or the special symbol “ε”. The symbol “ε” (epsilon) means “within a small value”, and for purposes of this specification it represents 1e-6 (so number±ε means within 1e-6 of the number). If there is a “±” symbol in the result, the expression (ABS(*Expression* – *First_Number*) < *Error_range)*=TRUE() **shall** be true, where *Expression* is the calculated expression. Any whitespace surrounding the “±” symbol is irrelevant.
- *Error:* The result **shall** be an error value, where NA is considered an error value. This is the same as ISERROR(*Expression*)=TRUE(). Note that this specification permits applications to have different kinds of error.
- *NA*: The result is NA. This is the same as ISNA(*Expression*)=TRUE().

Copyright © OASIS Open 2006. All Rights Reserved.

- =formula: The calculated expression **shall** be equal to the result of calculating OpenFormula "formula"; equality is determined using the "=" operator.
- { array-values }: The calculated expression is distributed over a set of cells (matching the positions in array-values), and each cell matches the given values.

Many test cases define an error range. Applications **should** produce correct values with significantly less error than the given error range, but **shall** produce results within at least that range to pass that test case.

TODO: There are a number of "to do" items that involve test cases, which have not yet been done because their completion was not required to create a normative specification.

Note: Test cases are designed to be automatic and only test one capability at a time, where possible. For example:

- The DATE() function is used to generate most date values, and similarly for TIME(), since they are portable everywhere. Lotus 1-2-3 has a DATEVALUE(), but its DATEVALUE() does not accept ISO 8601 format dates. Several applications do not automatically apply VALUE() or a similar function in their implied conversion to Number.
- Many applications interpret all text values as 0 when in a number context, including Lotus 1-2-3, Quattro Pro, and KSpread. Tests for conversions are in the conversion sections; the rest of the test cases are designed to *not* require conversion of text to a non-number.
- Floating-point values (including time values) are inexact in most implementations. Where this can be an issue, test cases subtract the expected value and use ABS() to ensure that the result is correct within a reasonable range. This is automatic when you use the "±" character.
- Not all implementations support inline arrays, so tests for a specific function that can be used in a non-array context should not use inline arrays. There are separate tests for support of inline arrays.

2.4 Test Case Data Set

In the test cases, the following spreadsheet values are presumed to be in the current sheet, as well as the sheets named Sheet1 and Sheet2:

	B	***C***
3	="7"	
4	=2	4
5	=3	5
6	=1=1	7
7	="Hello"	2005-01-31
8		2006-01-31
9	=1/0	02:00:00
10	=0	23:00:00
11	3	5
12	4	6

Copyright © OASIS Open 2006. All Rights Reserved.

13	2005-01-31T01:00:00	8
14	1	4
15	2	3
16	3	2
17	4	1

Note that B6 has the logical value TRUE(), B8 is a blank cell, and B9 is an error. C7 and C8 are dates, C9 and C10 are times, and B13 is a datetime value.

The named expressions FOUR and ΔΩ are assigned to cell C4 (the name ΔΩ is used to test for support of international characters).

At least the data above, including the named expression FOUR, is copied to a file in the current directory named 'openformula-testsuite.ods'.

The following is a trial test database; this range is assigned the name TESTDB. This database is purely for testing purposes; this data is not intended to be astronomically accurate:

	A	***B***	***C***	***D***	***E***	***F***	***G***	***H***	***I***
18	***TestID***	***Constellation***	Bright Stars	Northern	Abbrev	Decl	Next South	Date	Rev
19	1	Cancer	0	TRUE	Cnc	20	=B20	12 Mar 2005	13
20	=[.A19]*2	Canis Major	5	FALSE	Cma	5	=B27	3 Feb 2002	12
21	=[.A20]*2	Canis Minor	2	TRUE	Cmi	-20	= B24	8 Mar 2005	11
22	=[.A21]*2	Carina	5	FALSE	Car	-60		27 Mar 1991	10
23	=[.A22]*2	Draco	3	TRUE	Dra	75	=B31	5 Jul 1967	9
24	=[.A23]*2	Eridanus	4	FALSE	Eri	-29	=B29	23 Dec 1912	8
25	=[.A24]*2	Gemini	4	TRUE	Gem	20	=B19	6 Feb 1992	7
26	=[.A25]*2	Hercules	0	TRUE	Her	30	=B25	4 Jul 1934	6
27	=[.A26]*2	Orion	8	TRUE	Ori	5	=B21	8 Jan 1909	5
28	=[.A27]*2	Phoenix	1	FALSE	Phe	-50	=B22	28 Nov 1989	4
29	=[.A28]*2	Scorpio	9	FALSE	Sco	-40	=B28	22 Feb 2000	3

Copyright © OASIS Open 2006. All Rights Reserved.

30	=[.A29]*2	Ursa Major	6	TRUE	Uma	55.38	=B26	29 Mar 2004	2
31	=[.A30]*2	Ursa Minor	2	TRUE	Umi	70	=B30	13 Jul 1946	1

Notes:

- The TRUE and FALSE values in the database are actually =1=1 and =1=0 respectively (for maximum compatibility).
- Some of the dates are in the future. No dates are before 1904, for maximum compatibility.
- The last column is there solely to test reverse searches.
- The "Next south" field is for testing references. Notice that one of the references is blank.

The following are test criteria, for extracting data from the database:

	B	***C***	***D***	***E***	***F***	***G***	***H***	***I***
36	***Bright Stars***	***Northern***	***Constellation***	***Decl***	***Rev***	***Bright Stars***	***Date***	
37	4	TRUE	Ursa Major	< 0	<= 7	< 4	>1950-01-01	
38	<2	TRUE	Constellation			>= 8	CONSTELLATION	
39	>1	FALSE	Ursa				URSA MAJOR	

The following are especially helpful for some of the statistical tests:

	B	***C***	***D***	***E***	***F***	***G***	***H***	***I***
50	x1	x2	y	Value	x3	x4	x5	Probability
51	4	7	100	10.5	3	23	0	0.2
52	5	9	105	7.2	4	24	1	0.3
53	6	11	104	200	5	25	2	0.1
54	7	12	108	5.4	2	22	3	0.4
55	8	15	111	8.1	3	23		
56	9	17	120		4	24		
57	10	19	133		5	25		
58					6	26		
59					4	24		
60					7	27		

Test case data used with LEGACY.CHITEST:

	B	***C***	***D***	***E***	***F***	***G***	***H***
70	a1	a2	a3		e1	e2	e3
71	12	45	78		11	44	77

Copyright © OASIS Open 2006. All Rights Reserved.

72	23	56	89		22	55	88
73	34	67	98		33	66	99

Copyright © OASIS Open 2006. All Rights Reserved.

3 Formula Processing Model

This section describes the basic formula processing model: how expressions are calculated, when recalculation occurs, and limits on formulas.

3.1 Expression Calculation

Conceptually, formulas are recalculated from the “outside in”. Any formula is an expression that produces a result. An expression is calculated as follows:

1. If an expression is a constant number or string, that constant is returned
2. If it is a reference, the reference is returned. If a reference is to be displayed, the *value of the reference* is displayed, not the reference itself.
3. Otherwise, it is one or more operations or functions; in the case of operations, the highest-precedence operation not processed is processed first.
    a) The values of all argument expressions are computed, that is, formulas are normally “eagerly” evaluated. Exceptions to eager evaluation are noted in the function or operation's specification; in particular, the IF() function does not calculate the “else” parameter if the the condition is true, and does not calculate the “then” parameter if the condition is false. The CHOOSE() function does not calculate parameters other than the chosen. Function parameters **shall** act as if they had been computed in left-to-right order. Operators **should** act as if they had been computed in the order of precedence and associativity (so they are computed left-to-right for +, *, and so on, but right-to-left for the exponentiation operator ^).
    b) If any of the arguments of the function/operation are not of the correct type, the appropriate implicit conversion function is called to convert it to the correct type for the operator or function.
    c) The operation or function is then called with the resulting values of its arguments.

The above model only describes how recalculation appears to the end-user. Applications may, and typically do, optimize this process as long as the final results produce the same answer. For example, applications may parse a formula and translate it into some intermediate form (such as a byte code), which immediately descends to the “innermost” computation that needs to be calculated and then works out to the final result.

When a formula is computed, it is notionally provided a "context" as input. The context may include formula variables (including named ranges, document variables, fields, and so on), and/or additional function definitions that the formula can call. A formula may also be provided as input an ordered list of zero or more parameters (though the syntax for parameters is not given in this version of the specification). In an OpenDocument formula, this context also includes calculation settings (such as whether or not text comparisons are case-sensitive).

A formula may include calls to functions, which are normally provided the same context but with their own set of ordered parameters.

Any formula computes a single result, though that single result may actually be a set of values.

3.6 Basic Limits

Applications **should** avoid imposing arbitrary limits. Compliant applications which claim to support “basic limits” **shall** support at least the following limits:

Copyright © OASIS Open 2006. All Rights Reserved.

1. Applications **shall** support formulas up to at least 1024 characters long, as measured when in ODF interchange format not counting the square brackets around cell addresses, the "." in a cell address when the sheet name is omitted, namespace prefix, or the initial "=" symbol.

2. Applications **shall** support at least 30 parameters per function when the function prototype permits a list of parameters.

3. Applications **shall** permit strings of ASCII characters of up to 32,767 (2^15-1), at least.

4. Applications **shall** support at least 7 nesting levels of functions.

Thus, that portable documents **shall** stay within these limits to be portable.

Rationale: Excel 2003 supports 1,024 characters per formula, per Walkenbach "Excel 2003 Formulas" page 33, using its own UI representation. The measurement given above is carefully devised to mean the same thing in this case, since the representation is similar but not identical. The result is a reasonable measure of formulas that can be widely ported due to length. Most implementations are not this limited in size, anyway. Excel 2003 supports up to 30 parameters, according to OpenOffice.org's documentation on the .xls file format. Note that while string processing is not required to support international characters for basic, even basic *must* support international characters in variable names. The reason is simple: Many spreadsheet documents do not process strings at all, so having poor support for string-handling is actually sufficient for a large number of real-world use cases. They *would* need to support international characters in labels, but labels are outside the scope of this specification. However, named expressions (variables) are widely used in spreadsheets, and it would be unacceptable if the names were limited to only ASCII characters.

Test Cases:

Expression	***Result***	***Comment***
=SUM([.B4];[.B5];[.B4];[.B5];[.B4];[.B5]; [.B4];[.B5];[.B4];[.B5]; [.B4];[.B5]; [.B4];[.B5];[.B4];[.B5];[.B4];[.B5]; [.B4];[.B5];[.B4];[.B5]; [.B4];[.B5]; [.B4];[.B5];[.B4];[.B5];[.B4];[.B5])	75	Functions **shall** be able to take 30 parameters.
=[.B4]+[.B5]+[.B4]+[.B5]+[.B4]+[.B5]+[.B4]+[.B5]+[.B4]+[.B5]+ [.B4]+[.B5]+[.B4]+[.B5]+[.B4]+[.B5]+[.B4]+[.B5]+[.B4]+[.B5]+	1961	Formulas can be up to 1024 characters long, not counting the [..] around cell addresses, and not counting the "." in a cell address where the sheet name is not given.

Copyright © OASIS Open 2006. All Rights Reserved.

[.B4]+[.B5]+1111		
=LEN(REPT("x";2^15-1))	32767	Strings of ASCII characters can be up to 32767 characters (applications **should** support even larger text lengths)
=SIN(SIN(SIN(SIN(SIN(SIN(SIN(0)))))))	0	Support at least 7 levels of nesting functions.

Useful information about spreadsheets that was used to develop this specification included the following:

- *OpenOffice.org's Documentation of the Microsoft Excel File Format*. http://sc.openoffice.org. (This location has much other useful information).
- *Excel 2000 in a Nutshell: A Power User's Quick Reference*. Jinjer Simon. August 2000. O'Reilly.
- Ximian has a nice LXR setup that lets you easily surf the OpenOffice.org spreadsheet source code at http://ooo.ximian.com/lxr/source/sc/. Of particular interest is the Calc compiler.cxx file, see: http://ooo.ximian.com/lxr/source/sc/sc/source/core/tool/compiler.cxx.
- http://www.ccil.org/~cowan/OF provides the OpenOffice.org help documents for its functions in various formats. It is LGPL-licensed, because embedded in the OpenOffice.org implementation and the OpenOffice.org license states that "OpenOffice.org uses a single open-source license for the source code and a separate documentation license for most documents published on the website without the intention of being included in the product."
- http://www.gnome.org/projects/gnumeric/functions.shtml describes the Gnumeric functions. This text is believed to be licensed under the GPL.
- Many special tests of various spreadsheet implementations are at: http://www.dwheeler.com/openformula.
- http://www.mcs.vuw.ac.nz/~db/FishBrainWiki?Excel provides information on Microsoft Excel's display syntax
- http://rubyforge.org/projects/lxl LXL (Like Excel) is a mini-language that mimics Microsoft Excel formulas; written in Ruby
- Dan Bricklin (creator of VisiCalc, the original spreadsheet) is developing WikiCalc. This is an open source software web application that's a Wiki with spreadsheet capabilities added; it can locally or over a network. More info at: http://blogs.zdnet.com/BTL/?p=2141&part=rss&tag=feed&subj=zdblog

Copyright © OASIS Open 2006. All Rights Reserved.

Note: Excel 2003 is limited to 32,767 (2^15-1) ASCII characters per string. OpenOffice.org is limited to 65,535 (2^16-1) ASCII characters per string. For non-ASCII characters it is more complex to state limits; typical implementations use UTF-8, UTF-16, or UTF-32, so their limits would vary, and an implementation could even store text in a particular encoding along with an encoding marker. As a result, it's difficult to specify a limit for non-ASCII text lengths that makes sense. But with a large limit on ASCII text lengths, long lengths for non-ASCII text will tend to follow.

Copyright © OASIS Open 2006. All Rights Reserved.

4 Types

OpenFormula expressions, including formulas, always produce a value. Any table cell that can be referenced by an expression may be either empty (blank) or have a value. If it has a value, that value may be a constant or may itself be the result of calculating another formula. If a table cell has a constant value, it is exchanged using the format used to exchange such constant values. For example, see the OpenDocument standard for information on how to exchange constant table cell values (including constant date and time values).

Any value has a basic type. In addition, many functions require a type or a set of types with special properties (for example, a "Database" requires headers that are the field names); these specialized types are called *pseudotypes*. Some types (particularly Number) can be used in different ways, and can thus be formatted in different ways; these different uses are termed *subtypes*, since implementations typically track the specific subtype to heuristically determine the default formatting for newly-defined cells.

In OpenFormula a value **may** have one of the following basic types: *Text* (the string type), *Number*, *Logical* (the boolean type), *Error*, *Reference*, or *Array*. The Logical type **may** be implemented using the Number type, as described below, instead of being a distinct type. An application may choose to not implement complex numbers; if it implements complex numbers, they **may** be implemented using a distinct complex number type, or it **may** implement complex numbers using the Text or Number type. An implementation **may** provide other basic types, and it **may** have many specialized subtypes of these types beyond those mentioned in this specification. A portable document **shall** only depend on constructs defined in this specification (including *only* these types) to conform to it.

A reference may refer to a cell that does not contain a value; such a reference refers to an empty (aka blank) cell. An empty cell is neither zero nor the empty string, and an empty cell can be distinguished from cells containing values (including zero and the empty string) by some functions (such as ISBLANK and COUNTBLANK). An empty cell is not the same as an Error, in particular, it is distinguishable from the Error #N/A (not available). As discussed below, in many scalar contexts a reference to an empty cell is converted into 0 or the empty string, and in most sequences empty cells are automatically omitted. If the outermost expression is a reference to an empty cell, it **shall** be converted to the number 0. Thus, if cell A1 is empty, cell B1 computes “=[.A1]”, and cell C1 computes “=ISEMPTY([.B1])”, cell B1 will compute 0 and cell C1 will compute FALSE().

4.2 Number

A number is simply a numeric value such as 0, -4.5, or $1000. Numbers **shall** be able to represent fractional values (they **shall not** be limited to only integers). The "number" type **may** be displayed in many different formats, including date, time, percentage, and currency.

Typical implementations implement numbers as 64-bit IEEE floating point values and use the CPU's floating-point instructions where available (so intermediate values may be represented using more than 64 bits). However, implementations have great freedom in how they implement Number, and may use representations with a fixed bit length or a variable bit length. A cell with a constant numeric value has the number type.

Note that many formula creators are not sophisticated in their understanding of how computers determine their results. Many users, for example, do not understand computer floating point arithmetic models, and have no idea that many implementations use a base other than 10 (or what that would mean). In particular, many implementations use base 2 representations, with the

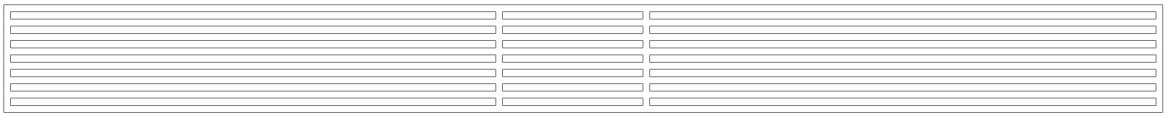
Copyright © OASIS Open 2006. All Rights Reserved.

result that value 0.1 can only be represented imprecisely (just as 1/3 can only imprecisely represented in a base 10 decimal representation). This problem applies to all uses of typical computing equipment, including nearly all programming languages, and this specification does not attempt to fully resolve the problem of unsophisticated users.

Rationale: Originally some efforts were expended to try to make formulas produce the "expected answer" for unsophisticated users. In particular, the equal-to operator for numbers matches imprecisely in many applications, because many users do not understand that (1/3)*3 on most implementations will produce a value close to one but not precisely equal to one. Originally there was a test to ensure that (1/3)*3 was equal to 1. The Gnumeric developers objected, on the grounds that *requiring* that equality be "sloppy" made it very difficult for sophisticated users to use spreadsheets to their full capabilities. In contrast, the function INT still requires that INT((1/3)*3) is 1, because if INT does not do so, many user's spreadsheets will not work as they expect. The expected answers of INT may not make numerical analysts happy, but users will get what appears (to them) to be wrong answers otherwise.

Implementations typically support many subtypes of Number, including date, time, datetime, percentages, fixed-point arithmetic, and arithmetic supporting arbitrarily long integers, and determine the display format from this. All such Number subtypes **shall** yield True for the ISNUMBER function. This specification does not require that specific subtypes be distinguishable from each other, or that the subtype be tracked, but in practice most implementations do such tracking because requiring users to manually format every cell appropriately becomes tedious very quickly. Automatically determining the most likely subtype is especially important for a good user interface when generating OpenDocument format, since some subtypes (such as date, time, and currency) are stored in a different manner depending on their subtype. Thus, this specification identifies some common subtypes and identifies those subtypes where relevant in some function definitions, as an aid to implementing good user interfaces. Many applications vary in the subtype produced when combining subtypes (e.g., what is the result when percentages are multiplied together), so unless otherwise noted these are unspecified. Typical implementations try to heuristically determine the "right" format for a cell when a formula is first created, based on the operations in the formula. Users can then override this format, so as a result the heuristics are not important for data exchange (and thus outside the scope of this specification).

In OpenDocument, unless otherwise noted, the `office:value-type` of number values is `float`; if the computed value is stored, it is stored in the attribute `office:value`. The selection of a subtype is decided by the display format, as discussed below.



4.4 Logical (Boolean)

4.5 Error

An error is one of a set of possible error values. Implementations may have many different error values, but one error value in particular is distinct: #N/A, the result of the NA() function. Users may choose to enter some data values as #N/A, so that this error value propagates to any other formula that uses it, and may test for this using the function ISNA().

Functions and operators that receive one or more error values as an input **shall** produce one of those input error values as their result, except when the formula or operator is specifically defined to do otherwise.

In an OpenDocument document, if an error value is the result of a cell computation it **shall** be stored as if it was a string. That is, the `office:value-type` of an error value is `string`; if the computed value is stored, it is stored in the attribute `office:string-value`. Note that this

Copyright © OASIS Open 2006. All Rights Reserved.

does not change an error into a string type (since the error will be restored on recalculation); this simply enables applications which cannot recalculate values to display the error information.

Note: This treats errors as strings, instead of as their own special type, when stored. Storing as strings is sufficient, since this is solely for the purpose of displaying results by applications that don't implement spreadsheets. A recalculation will restore the error to a full and more detailed error type.

Note: We keep maximum flexibility by simply asserting that there are error values without saying what they are other than NA. In practice, users generally don't care which errors are which (other than an NA) in most cases, and implementations do different things in different cases.

Note: Excel 2003 has the following error values (with ERROR.TYPE values), according to Walkenbach 2003, page 49 and Simon's "Excel 2000 in a Nutshell" page 527:

- #DIV/0! (2) - Attempt to divide by zero, including division by an empty cell.
- #NAME? (5) - Unrecognized/deleted name.
- #N/A (7) - NA. Lookup functions which failed and NA() return this value.
- #NULL! (1) - Intersection of ranges produced zero cells.
- #NUM! (6) - Failed to meet domain constraints (e.g., input was too large or too small)
- #REF! (4) - Reference to invalid cell.
- #VALUE! (3) - Parameter is wrong type.

4.9 Pseudotypes

Many functions require a type or a set of types with special properties, and/or process them specially. For example, a "Database" requires headers that are the field names. These specialized types are called *pseudotypes*.

4.9.1 Scalar

A *Scalar* value is a value that has a *single* value. A reference to more than one cell is *not* a scalar (by itself), and **shall** be converted to one as described below. Similarly, an array with more than one element is not a scalar. The types Number (including a complex number), Logical, and Text are scalars.

Copyright © OASIS Open 2006. All Rights Reserved.

5 Expression Syntax

Any application **shall** support the syntax as described below, except for the noted exceptions, for the capabilities it supports. Any reading application **shall** be able to read *at least* the syntax defined in this section (it may accept various extensions) for the capabilities it supports. Any writing application **shall** generate this format for the capabilities it supports when a user creates a document that can comply with it; this is obviously not possible for some application-specific extensions.

Note that being able to read the syntax is not necessarily the same as being able to recalculate its results: expressions may be syntactically well-formed yet semantically meaningless. In addition, some applications might not support some more advanced capabilities such as inline arrays, inline arrays with arbitrary (non-constant) expressions, cell concatenation, external references to arbitrary cell values, or subtables.

This syntax is defined using the BNF notation of the XML specification, chapter 6 [XML10]. Note that each syntax rule is defined using "::=".

Note that formulas are typically embedded inside an XML document. When this occurs, various characters (such as "<", ">", "", and "&") **shall** be escaped, as described in section 2.4 of the XML specification [XML10]. In particular, the less-than symbol "<" is typically represented as “<”, the double-quote symbol as “"”, and the ampersand symbol as “&” (alternatively, a numeric character reference can be used).

An example of the expression syntax is “=5+2*SUM([.A1:.B2]; 4)+ 8<9”. In an XML document, this might be represented as ...table:formula=“=5+2*SUM([.A1:.B2]; 4)+ 8<9”...

5.2 Basic Expressions

Formulas are simply an '=', followed by an optional “forced recalculate” marker “=”, followed by an expression. If the second '=' is present, this formula is a "forced recalculation" formula. Typical implementations optimize what is recalculated, and typically only recalculate what is displayed and the values they depend on (transitively). If the formula is marked as a "forced recalculation" formula, then it **should** be recalculated whenever one of its predecessors it depends on changed, even if it does not appear to be necessary for display:

```
Formula ::= '=' ForceRecalc? Expression
ForceRecalc ::= '='
```

Rationale: User-defined functions can have side-effects, though they are not recommended. Having a ForceRecalc marker allows users to force recalculation in cases the dependency analysis of an application would miss. Also, some functions can refer to values without it being clear to a dependency analysis which ones it depends on. This is a specialized capability, and most documents will not need to use forced recalculation.

Note: Spreadsheet implementations decide what to recalculate, and may decide to consider other formulas as if they were "forced recalculate" formulas based on the functions used in them.

The primary component of a formula is an `Expression`. Formulas are composed of `Expression`s, which may in turn be composed from other `Expression`s.

```
Expression ::= Number |
      String |
      Array |
      PrefixOp Expression |
```


Copyright © OASIS Open 2006. All Rights Reserved.

```
      Expression PostfixOp |
      Expression InfixOp Expression |
      '(' Expression ')' |
      FunctionName '(' ParameterList ')' |
      Reference |
      QuotedLabel |
      AutomaticIntersection |
      NamedExpression |
      Error
SingleQuoted ::= "'" ([^'] | "''")+ "'"
```

The syntax defines rules for literal numbers and literal strings, but it does not define a rule for literal logical constants. For logical constants, TRUE() and FALSE() are used.

Note: The test cases are in the specific sections for each alternatives, rather than here.

Note: Several different syntactic units in an expression can have the representation '...', with possible embedded pairs of single quotes. This defined here as SingleQuoted; SingleQuoted is not actually referenced here, but in several different syntactic units below. These units are disambiguated by the non-whitespace character following it. Thus, in, 'Sheet1'.Name1, Sheet1 is a sheet, while in 'http://example.com/'#'Sheet1'.xyz, the first part is an IRI. A typical lexer can treat all uses of '...' as the same syntactic unit, and then let a higher-level parser handle the different cases, so that single syntactic unit (SingleQuoted) is documented here.

5.3 Constant Numbers

Constant numbers are written using '.' dot as the decimal separator. Optional "E" or "e" denotes scientific notation. Syntactically, negative numbers are simply positive numbers with a prefix "-" operator; implementations **may** and typically do optimize this operator by performing the negation and storing the negative number directly. A constant number, as defined by this syntax, **shall** be considered to be type Number.

```
Number ::= StandardNumber |
   '.' [0-9]+ ([eE] [-+]? [0-9]+)?
StandardNumber ::= [0-9]+ ('.' [0-9]+)? ([eE] [-+]? [0-9]+)?
```

Readers **should** be able to read the `Number` format, which accepts a decimal fraction that starts with decimal point '.', without a leading zero. Compliant writers **shall** write numbers only using the `StandardNumber` format, which requires a leading digit, and **shall not** write numbers with a leading '.'.

Note: Application implementors need to remember that when reading/writing numbers, use the “C” locale, e.g., POSIX setlocale(LC_NUMERIC, "C") or equivalent. The user's locale may use a different syntax, such as using “,” as the decimal separator. Note that leading - and + signs are allowed as unary prefix operators, and “%” as a postfix operator; syntactically they are operators, though applications should optimize them away.

Rationale: Writers are required to write a leading digit, because future versions of this specification might not include a requirement to accept numbers with a leading “.”.

Negative numbers are treated as a number with a prefix “-” operator. This makes the BNF very clean, and this way precedence is easy to specify correctly. Other language definitions, such as the one for Ada95, do this as well. There is a risk of incorrectly conflicting with the precedence rules if you try to handle "-" in the number lexical processing; it's just too easy to get things wrong. Microsoft Excel, OpenOffice.org, and most other spreadsheet applications treat prefix “-” with a uniform precedence (whether or not it precedes a constant number of an expression). However,

Copyright © OASIS Open 2006. All Rights Reserved.

there is at least one application (Microsoft Works) where unary "-" has different precedence, depending on whether or not it's in front of a constant number, according to http://www.burns-stat.com/pages/Tutor/spreadsheet_addiction.html. Since it's obvious that implementations can make this mistake, writing the BNF in this clear way (without making negative numbers a special case) prevents this misunderstanding entirely.

Test Cases:

Expression	*Result*	*Comment*
=56.5	56.5	Numbers use “.” as the decimal separator; trivial fractions supported.
=.5	0.5	Readers accept initial “.” for constant numbers, but should not write them.
=550E-1	55	Exponents can be negative
=550E+1	5500	Exponents can be positive
=56e2	5600	Exponents can have no sign (+ assumed) and lowercase “e” is okay

5.5 Operators

Operators are functions of one or more parameters, but with a special syntax. There are two predefined prefix operators: prefix plus (a no-op) and prefix minus (negation). There is one predefined postfix operator, percentage (%), which divides the preceding expression by 100. There are also a number of infix operators, written using traditional infix notation:

```
PrefixOp ::= '+' | '-'
PostfixOp ::= '%'
InfixOp ::= ArithmeticOp | ComparisonOp | StringOp | ReferenceOp
ArithmeticOp ::= '+' | '-' | '*' | '/' | '^'
ComparisonOp ::= '=' | '<>' | '<' | '>' | '<=' | '>='
StringOp ::= '&'
```

There are three predefined reference operators: reference intersection (which some user interfaces display as a space), reference concatenation (which some user interfaces display as the function parameter seperator), and range. The result of these operators may be a 3 dimensional range, with front-upper-left and back-lower-right corners, or even a list of such ranges in the case of cell concatenation. Note that since they are defined as general operators, either or both sides may be the output of a function, and not a constant cell reference. Note that many applications do *not* support the cell concatenation operator:

```
ReferenceOp ::= IntersectionOp | ReferenceConcatenationOp | RangeOp
IntersectionOp ::= '!'
ReferenceConcatenationOp ::= '~'
RangeOp ::= ':'
```

The operators have the following associativity and precedence, from highest to lowest precedence:

Copyright © OASIS Open 2006. All Rights Reserved.

Table 1 - Operators

Associativity	***Operator(s)***	***Comments***
left	:	Range.
left	!	Reference intersection ([.A1:.C4]![.B1:.B5] is [.B1:.B4]). Displayed as the space character in some implementations.
left	~	Reference union. Displayed as the function parameter separator in some implementations.
right	+,-	Prefix unary operators, e.g., -5 or -[.A1]. Note that these have a different precedence than add and subtract.
left	%	Postfix unary operator % (divide by 100). Note that this is legal with expressions (e.g., [.B1]%), it can be duplicated (1%%), and it does *not* change the meaning of other operations such as "+".
left	^	Power (2 ^ 3 is 8).
left	*,/	Multiply, divide.
left	+,-	Binary operations add, subtract. Note that unary (prefix) + and - have a different precedence.
left	&	Binary operation string concatenation. Note that unary (prefix) + and - has a different precedence. Note that "&" **shall** be escaped when included in an XML document
left	=, <>, <, <=, >, >=	Comparison operators equal to, not equal to, less than, less than or equal to, greater than, greater than or equal to

Note that prefix “-” has a higher precedence than “^”, that “^” is left-associative, and that reference intersection has a higher precedence than reference union.

Prefix “+” and “–“ are defined to be right-associative. However, note that typical applications which implement at most the operators defined in this specification (as specified) **may** implement them as left-associative, because the calculated results will be identical.

Precedence can be overridden by using parentheses, so "=2+3*4" computes to 14 but "=(2+3)*4" computes 20. Implementations **should** retain "unnecessary" parentheses and white space, since these are added by people to improve readability.

Note: There are no test cases in this section, because the test cases are provided in the text about the individual operators. This was done to eliminate duplication in this specification, and also because some applications may choose to only implement a subset (e.g., not implementing cell concatenation).

Implementations' user interfaces may display these operators differently or with a different precedence, but when writing or reading formulas they must use the precedence rules here.

Rationale: The above is the conventional set of operators for most spreadsheet applications, along with their conventional precedence and associativity. This format is intentionally similar to traditional presentations of spreadsheet formulas, which reduces the likelihood of error or misunderstanding. For example, Microsoft Excel documentation at http://support.microsoft.com/kb/25189/EN-US/) claims that the precedence is (from highest to lowest) Range, Intersection, Union in that order. Walkenbach gives Microsoft Excel 2003's precedence levels as (lowest to highest, note that the book gives the reverse order) comparison (such as "="), "&", "+" and "-", "*" and "/", "^", "%", and unary "-" (negation). (Walkenbach, 2004, pg

Copyright © OASIS Open 2006. All Rights Reserved.

38). Prefix "-" and "+" are right-associative, not non-associative, because "--[.B3]" is legal (and it converts B3 to a number, so it can have an effect). Cell address intersection has an even higher precedence than “-” so that a unary minus in front of an intersection will work correctly.

Parentheses are not normally documented as an “operator” in spreadsheet applications, so they are not considered operators here as well. Instead they are simply part of expression syntax. If parentheses were considered to be operators with a precedence, they would have a precedence higher than any of the operators here.

See the infix operator “^” text for the rationale for its left-to-right associativity, and why prefix “-” has a higher precedence than infix “^”. Not all applications have historically done this, but many have, and the group agreed that it needed to be specified, and not left as being implementation-dependent.

Although the user interface is not specified here, the representation is intentionally chosen so that formulas can "round trip" to this format and back without loss using typical formula representations. For example, it would be possible to replace some or all uses of "%" with "/100", and to replace "^" with POWER(), but this would cause the display format to change once it was saved and reloaded.

5.6 Functions and Function Parameters

Functions are called by giving their name, followed by parentheses surrounding a list of parameters. Parameters are separated using the semicolon (;) character:

```
FunctionName ::= Identifier
Identifier ::= LetterXML (LetterXML | DigitXML |
      '_' | '.' | CombiningCharXML)*
```

Where LetterXML, DigitXML, and CombiningCharXML are Letter, Digit, and CombiningChar as they are defined in [XML10].

Applications' user interfaces **may** (and often do) display a different function name in their user interface; for example, they may translate function names to the current locale, omit application prefixes, or replace the names with arbitrary other names. But conforming applications **shall** use the names as defined by this specification, and **shall not** use these other names, when storing and reading the names of functions defined by this specification.

Function names are case-insensitive, so SUM and Sum are the same function. However, traditionally spreadsheet function names are written in all upper case, so this tradition is followed in this specification. Implementations **should** write function names in all upper case when saving to a document.

Some functions always produce the same value, and are thus actually constants. These include PI(), TRUE(), and FALSE(). Implementations **should** optimize these, where appropriate, in at least the same way they optimize other constants.

Note: User interfaces may, of course, display a different parameter separator. The comma (“,”) is a common alternative separator, though note that using comma as a parameter separator interferes with its use as a decimal or “thousands” separator.

User-defined function names may use an arbitrary Identifier, but the names of the functions predefined by this specification use the stricter format [A-Za-z] [A-Za-z0-9_.]*.

Rationale: Function names are often displayed in all upper case by both implementations, showed this way in documentation, and saved in uppercase, owing to the influence of the first spreadsheet program (VisiCalc). By saying implementations **should** save in all uppercase, this increases the likelihood that implementations can trivially write back formulas and, if the formulas are unchanged, produce an identical result. This can be helpful to systems that depend on

Copyright © OASIS Open 2006. All Rights Reserved.

detecting differences through simple text comparisons – having this convention minimizes the number of unnecessary differences in the text.

Some applications, such as Excel and Gnumeric, use the "," as the function parameter separator in their user interface. Others, such as OpenOffice.org and Lotus 1-2-3, use ";" instead. Many locales use "," as the decimal separator; using the semicolon as the parameter separator eliminates confusion and the risk of incorrect implementation. What the user interface uses is not relevant to this specification.

Note: Microsoft Excel accepts empty parameters in any position. OpenOffice.org 1.1.3 did not. Typical implementations will have many built-in functions, and most implementations also support one or more ways to create user-defined functions.

Function calls **shall** be given a parameter list, though it may be empty. An empty list of parameters is considered a call with 0 parameters, not a call with one parameter that happens to be empty. Thus, TRUE() is syntactically a function call with 0 parameters. It is syntactically legitimate to provide empty parameters, though functions are not required to accept empty parameters unless otherwise noted:

```
ParameterList ::= /* empty */ |
       Parameter (Separator EmptyOrParameter )* |
       Separator EmptyOrParameter /* First param empty */
       (Separator EmptyOrParameter )*
EmptyOrParameter ::= /* empty */ | Parameter
Parameter ::= Expression
Separator ::= ';'
```

Test Cases:

Expression	*Result*	*Comment*
=TRUE()	True	Syntactically this is a zero-parameter function call, *not* a one-parameter function call whose parameter happens to be empty. Implementations will typically treat TRUE() as a constant.
=ABS(4)	4	One parameter
=MAX(2;3)	3	Two parameters – note that ";" is the separator, not ","
=IF(FALSE();7;8)	8	Simple if, three parameters
=IF(FALSE();7;)	0	Empty parameter for "else" parameter is considered 0 by IF

Note: If there is a need for passing exactly one parameter that is empty, a function like EMPTYPARAM() could be defined for representing an empty parameter. However, the committee has not yet identified any need for such a thing, and saw no reason to invent the unnecessary. If it's needed in the future, it could be added.

5.8 References

#TODO JH: support reference to fields/variables, but not to cells#

References refer to a specific cell or set of cells. Some functions return values of the Reference type. This is the syntax for a constant reference:

```
Reference ::= '[' Source? RangeAddress ']'
RangeAddress ::=
```


Copyright © OASIS Open 2006. All Rights Reserved.

```
 SheetLocator "." Column Row (':' SheetLocator "." Column Row )? |
 SheetLocator "." Column ':' SheetLocator "." Column |
 SheetLocator "." Row ':' SheetLocator "." Row
SheetLocator ::= SheetName ("." SubtableCell)*
SheetName ::= QuotedSheetName | '$'? [^\]\. #$']+ | /*empty */
QuotedSheetName ::= '$'? SingleQuoted | Error
SubtableCell ::= ( Column Row ) | QuotedSheetName
Column ::= '$'? [A-Z]+
Row ::= '$'? [1-9] [0-9]*
Source ::= "'" IRI "'" "#"
CellAddress ::= SheetLocator "." Column Row /* Not used directly */
```

References always begin with '['; this immediately disambiguates cell addresses from function names and named expressions. `SheetName`s include single-quote(') characters by doubling them and having the entire name surrounded by single-quotes. Column labels **shall** be in uppercase. The syntax above supports whole-row and whole-column references, but they **shall** always have a range marker, e.g. [.A:.A] would be a reference to the whole first column and [.1:.1] a reference to the whole first row of the current sheet. The optional "$" marking in sheet names, `Column`, and `Row` is used to identify absolute values; without the marking they are relative. A reference, as defined by this syntax, **shall** be considered to be type Reference. Note that most functions, when given a reference, examine the values pointed to by the reference (and not the reference value itself).

Columns are named by a sequence of one or more uppercase letters A-Z (U+0041 through U+005A). Columns are named A, B, C, ... X, Y, Z, AA, AB, AC, ... AY, AZ, BA, BB, BC, ... ZX, ZY, ZZ, AAA, AAB, AAC, AAZ, ABA, ABB, and so on.

A reference with an explicit row or column value beyond the capabilities of the application **shall** be computed as an Error, and not as a reference. Authors of portable documents **may** use whole-row and whole-column references, such as [.1:.1] or [.A:.A], to facilitate updating a document to large sizes.

Note that references can include a single embedded ":" separator. Where possible, applications **should** use references with embedded ":" separators inside the [..] markers, instead of the general-purpose ":" operator, when saving files, and where there is a choice of cells to join, and application **should** choose the leftmost pair. Thus, an application **should** write "[.A1:.A3]" instead of "[.A1]:[.A3]". They are semantically identical, but this increases the likelihood that an unchanged function can be read and written back out unchanged.

Source location's IRI is described in RFC3987, Internationalized Resource Identifiers (IRIs), based on RFC3986, Uniform Resource Identifier (URI): General Syntax. Implementations **should** support absolute IRIs (URLs are IRIs too). Implementations **should** support relative IRIs, which can be distinguished because they do not begin with [A-Za-z]+ ":". Relative IRIs are formed according to section 6.5 of RFC3987, respectively section 4.2 of RFC3986. Applications **should** always use a "./" prefix when writing a relative IRI, since this is unambiguous. Applications **should** support the *file* scheme (file:// prefix).

Applications **may** support a variety of IRI/URI/URL schemes (such as "http:"), but they should beware of the security ramifications. In particular, an attacker could use this ability to note when someone views a file if this is automatically loaded, or possibly even use automatic loading as a way to send malicious data back to the spreadsheet creator. Implementations **should** ask the user for confirmation before loading data for the first time from a particular external data source.

Applications **may** omit support for subtables (and typically do), unless they claim conformance to another part of this specification that requires such support.

Copyright © OASIS Open 2006. All Rights Reserved.

Test Cases:

Expression	*Result*	*Comment*
=[.B4]	2	Simple reference
=[.B4]	2	Absolute reference
=[.$B4]	2	Partly absolute reference
=[.B$4]	2	Partly absolute reference
=SUM([.B4:.B5])	5	Simple range
=[Sheet1.B4]	2	Explicit sheet name
=['Sheet1'.B4]	2	Explicit sheet name, quoted
=SUM([Sheet1.B4:Sheet1.B5])	5	Simple range with explicit Sheet name
=SUM([Sheet1.B4:Sheet2.C5])	28	Simple 3D range, naturally with explicit sheet names
['./openformula-testsuite.ods'#'Sheet1'.B4]	2	External reference to local IRI. This is a **should**, not a **shall**, so an application can pass this section without passing this test case.

Note: The text here is consistent with OpenDocument's definitions of cellRangeAddress and cellAddress. The “$” marking make a cell address absolute, otherwise it is relative (this has an effect when formulas are copied). OpenDocument also defines a cellRangeAddressList (space-separated ranges) which is used by several OpenDocument constructs. OpenFormula does not use the cellRangeAddressList grammar; cell concatenation can do the same thing when this is necessary.

Referencing a sheet that was deleted is typically turned into an error, such as #REF!.

A SheetName may be quoted, so a reference that begins with ['...' may be describing a Source or a SheetName. However, the next non-whitespace character eliminates the ambiguity – if it is “#”, it is the Source.

Typical implementations will consider Source as SingleQuoted; the text is shown this way to emphasize that it is an IRI that is expected between the single quote marks.

Typical spreadsheet displays will often *not* display or require input of the square brackets.

Note that user interfaces may omit the surrounding [...] of a reference.

Rationale: Cell addresses in OpenFormula begin with "[" and end with a "]"; this makes parsing simpler, faster, and more reliable. Cell addresses are specified in A1 notation, not an R1C1 notation, as required by OpenDocument 1.0 section 8.1.3 subsection "Formula". Not using R1C1 notation can in some cases cause an increase in compressed file size (because copied formulas are shown differently and thus do not compress as well). However, A1 notation is much easier for humans to understand, so using A1 format is likely to increase reliability and debuggability (because it is more likely to be correctly generated and interpreted). Where this is a serious problem, array formulas should be used instead, which can achieve the same results. Both R1C1 and A1 could be allowed, but then different spreadsheets are likely to generate different characters for the same cell, and thus create "differences" that do not exist even in simple spreadsheets. Only uppercase characters are allowed, for the same reason. Column labels shall be in uppercase; this makes it easier to distinguish between column labels and many named

Copyright © OASIS Open 2006. All Rights Reserved.

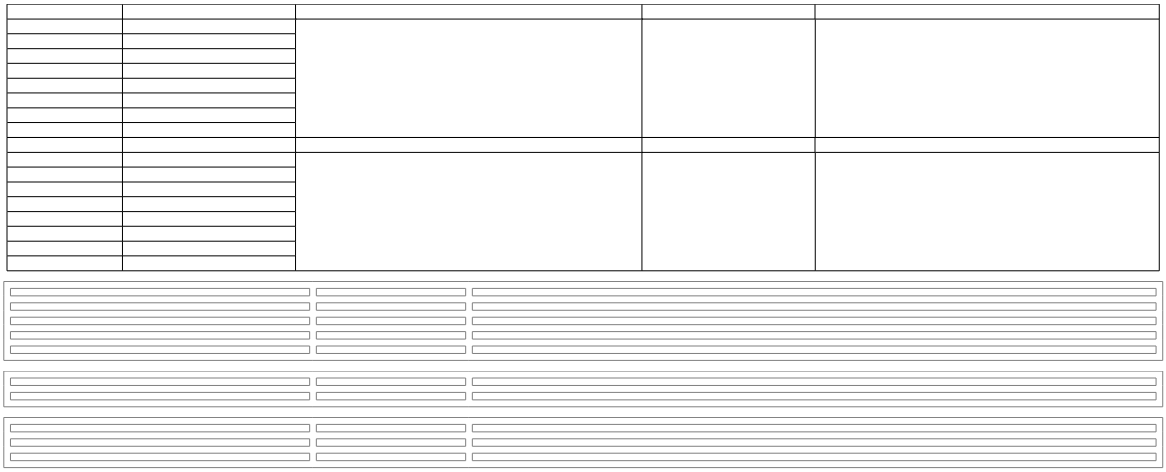
expressions, and increases the likelihood that functions can be read and written unchanged by different applications.

The above notation supports selectors for whole-rows and whole-columns. Applications which cannot directly support them can try to fake them by translating them into a large range, but should not do so, because they mean different things. Spreadsheet applications of the future are likely to support larger and larger limits; whole-row and whole-column notations do not change their meaning in such cases, while limited-range expressions do. We directly support these selectors so that large spreadsheets of the future would not silently fail.

In the syntax rules above, it was easier to show the rules for RangeAddress by repeating SheetLocator each time as well as the built-in range operation. Yet people often discuss individual “Cell Addresses” in conversation, meaning something that does *not* include a range of any kind. Therefore, an explicit rule for the syntax of CellAddress is given, even though it is not directly used by the syntax rules for formulas (it is subsumed by RangeAddress).

Note that the above can be easily expanded to support subtable addressing, e.g., [.B2.C3], though it does not require applications to support it. OpenDocument 1.0 section 8.3.1 discusses subtables, and requires the syntax A1.B2 when addressing subtables in those contexts (which are not formulas).

Note that the above syntax requires “duplication” of the prefix if you are asking for a range, e.g., [Sheet1.B2:Sheet2.C3], or [.B2.C3:.B2.D4]. This duplication is slightly inconvenient when you simply want the same sheet location, but has the advantage of being extremely flexible, and is very explicit in terms of what is desired, so this is considered to be the better approach. Note the careful definition of the syntax for subtables; the syntax has to differentiate between sheet names in the earlier parts and row/column addresses, and the names of named expressions and function calls can also include “.”.



5.12 Constant Errors

#TODO JH: necessary?#

Constant error values may be stored in the text of the formula itself, though this is not recommended as a general practice. It is discouraged because applications are not required to have the same kinds of errors, so the resulting formulas are generally not portable. It also not typically necessary, for example, creators of portable documents **may** use the NA() function for the NA error value instead of using an inline constant error value, and **may** use “=1/0” to portably represent a division by zero error. A constant error, as defined by this syntax, **shall** be considered to be type Error.

Copyright © OASIS Open 2006. All Rights Reserved.

However, there are cases where constant errors in a formula are valuable. For example, formulas may reference sheets that have been deleted since the formula was created, and as a result **may** have an error value for their `SheetName`.

Inline error constants **shall** have the following syntax:

```
Error ::= '#' [A-Z0-9]+ ([!?] | ('/' ([A-Z] | ([0-9] [!?]))))
```

Specific error values are indicated by an identifier. The only portable constant error value is "#N/A", which represents the NA() inline error value. Applications **should** use the following identifier names when they intend to represent that particular error and there is no more specific error that they are able to represent:

Table 4 - Recommended Constant Errors (when no more specific information is available)

Name	***Comments***
#DIV/0!	Attempt to divide by zero, including division by an empty cell. ERROR.TYPE of 2
#NAME?	Unrecognized/deleted name. ERROR.TYPE of 5.
#N/A	Not available. ISNA() applied to this value will return True. Lookup functions which failed, and NA(), return this value. ERROR.TYPE of 7.
#NULL!	Intersection of ranges produced zero cells. ERROR.TYPE of 1.
#NUM!	Failed to meet domain constraints (e.g., input was too large or too small). ERROR.TYPE of 6.
#REF!	Reference to invalid cell (e.g., beyond the application's abilities). ERROR.TYPE of 4.
#VALUE!	Parameter is wrong type. ERROR.TYPE of 3.

An unknown constant error value **shall** be mapped into an error value supported by the application when read (e.g., the application's equivalent of #NAME?), though an application **may** warn the user if this has or will take place. It is desirable to preserve the original specific error name when writing an error constant back out, where possible, but applications **may** write a different error value for a formula than they did when reading it for errors other than #N/A. Whitespace **shall not** be included in an Error name.

Applications **should** use a human-comprehensible name, not simply a numeric id, for constant error values they write. This aids interoperability with other applications that may use a different error numbering system.

Portable documents **should not** use inline error values other than #N/A, as error values are not necessarily portable between applications.

Test Cases:

Expression	***Result***	***Comment***
=#N/A	NA	Another way to write "NA"
=ISERROR(#N/A)	True	This tests to ensure that the parser can find the "end" of the error
=#DIV/0!	Error	This tests for handling "/0!" correctly
=ISERROR(#DIV/0!)	True	
=#NAME?	Error	We'll test all the "well-known" codes

Copyright © OASIS Open 2006. All Rights Reserved.

=ISERROR(#NAME?)	True	
=#NULL!	Error	
=ISERROR(#NULL!)	True	
=#NUM!	Error	
=ISERROR(#NUM!)	True	
=#REF!	Error	
=ISERROR(#REF!)	True	
=#VALUE!	Error	
=ISERROR(#VALUE!)	True	
=#UNKNOWNERRORCODE!	Error	Applications **shall** be able to read an unknown error value generated by another application, and map it into one of their own error codes. If it has such a code, “#NAME?” or equivalent would be appropriate
=ISERROR(#UNKNOWNERRORCODE!)	True	Unknown error codes still need to be processed as errors

Rationale: There was significant discussion on whether '/', '!', and '?' should be allowed in a name, since it would simplify parsing in some circumstances if they were forbidden. This has been kept, for the simple reason that spreadsheet applications which *do* share error names tend to use these names, with these spellings. And it does not seem hard to parse them this way, so there doesn't seem to be a strong reason to “simplify” these. Originally we had “Error ::= '#' [^\. #']+”, but this was an extremely loose syntax likely to cause trouble – it'd be too easy to miss the “end” of the error term. And while there were some original concerns about making the lexing of this simple, that's been resolved too. We want applications to be free to have a different and more refined error system, though, so we want a regular expression that will enable them to write out specific forms (so other applications will know where the error name ends).

Rationale: Since spreadsheet implementations may have additional or more specific types of errors, the notion of error has been generalized. Many implementations share these error values, however, so these representations are recommended so sharing is possible where appropriate.

The recommended names are used by many implementations.

Note: A trivial lexer cannot simply parse anything beginning with “#” as an error, because “#” can also occur as part of a Source identifier. One solution is to lex simply “#” as a character; if a parser receives “#” where it could be an Error value, it could switch the lexer to a different mode solely for reading the text of the error constant (or just call a different lexer), then switch back. Modern lex tools, such as flex, can do this easily.

Copyright © OASIS Open 2006. All Rights Reserved.

5.14 Whitespace

For calculation purposes, whitespace is generally ignored unless it is inside the contents of string constants or text surrounded by single quotes. Specifically, applications **shall** ignore any whitespace characters before and/or after any operators, constant numbers, constant strings, constant errors, inline arrays, parentheses used for controlling precedence, and the closing parenthesis of a function call. Whitespace **shall** be ignored following the initial equal sign(s). Whitespace **shall** be ignored just before a function name, but whitespace **shall not** separate a function name from its initial opening parentheses. Whitespace **shall not** be used in the interior of a terminating grammar rule (a rule that references no other rule other than character sets, internally or externally-defined), unless specifically permitted by the terminating grammar rule, since these rules define the lexical properties of a component. As a result, applications **shall not** write formulas with whitespace embedded in any unquoted identifier, constant number, or constant error. Thus “= 3 . 5 + 3” is not a legal formula, because the syntax for Number does not permit interior whitespace, but “= 3.5 + 3 ” is a legal formula. Note that “= 3 %” is also legal. Applications **shall** consider the following characters as whitespace characters: space (U+0020), tab (U+0009), newline (U+000A), and carriage return (U+000D).

An embedded line break **shall** be represented by a single newline character (U+000A), *not* by a carriage return-linefeed pair. When embedded in an XML document the newline character is typically represented as “&#0A;”.

Applications **should** retain whitespace entered by the original formula creator and use it when saving or presenting the formula, and **should not** add additional whitespace unless directed to do so during the process of editing a formula.

Test Cases:

Expression	*Result*	*Comment*
= 3.5 + 3	6.5	Whitespace permitted
= (2 + 3) * 5	25	Whitespace permitted around ordinary parentheses used for grouping
= 300 %	3	Percent is not special; it can be surrounded by whitespace too

Rationale: Whitespace (including newlines) should be retained so it can be used to make complex formulas easier to understand. Retaining is only a “should” because this can be burdensome to implement in some implementations, and because changing whitespace outside of a string constant and single-quoted value does not change the calculated result of an expression.

In some applications, such as Microsoft Excel, the space is used as an operator (such as the intersection operator) and also as ignorable whitespace. This is very confusing, making it difficult to parse and risky to use. The OpenFormula exchange format uses "!" instead of whitespace for the intersection operation, which eliminates this painful and unnecessary problem.

Although the exchange format can handle it, some Excel-like display formats could have difficulties with whitespace that separates a function name from its initial opening parentheses. As a small concession for this common case, that particular combination is not permitted.

Note: For purposes of this specification, the “user interface” is the textual representation of formulas as presented to end-users. This **may** be the same as the exchange syntax, but it need not be and often won't be. The exchange syntax is optimized for unambiguous exchange between

Copyright © OASIS Open 2006. All Rights Reserved.

different applications (which may implement different capabilities) and parsing at high speed (disambiguation markers aid in this).

In contrast, user interfaces often emphasize brevity, with some attempt to “automatically do what the user meant” and/or provide an interface the user happens to be familiar with. Today's applications tend to have different user interfaces, and at least one application (Corel Quattro Pro) allows the user interface to be selected and changed dynamically. User interfaces often differ on details such as what is the formula parameter separator (semicolon or comma), do function calls need an “@” prefix or not, the character for reference intersection (such as space or “!”), reference union (“~” or “,”), range (“:” or “..”), the equality (“=” or “==”) and inequality (“<>” or “!=”) operators, and how to refer to external documents, other sheets, or named expression identifiers. For document exchange these differences are irrelevant; users can simply choose the application that provides the user interface they prefer.

However, some implementors may want to have a “reasonable” UI intentionally similar to the exchange format, but where users do not need to type in various markings such as [...] around cell references. The text below demonstrates a sample user interface that is intentionally similar to the exchange format. This is provided as an aid to implementors who wish to create a simple user interface that easily works with the exchange format, since it is an important use case. This analysis also helped in examining the syntax given above, to eliminate potential problems in it.

More generally, a user interface will accept text and determine what is being input. For our purposes, we will assume that all functions begin with an initial character of “=”. Without an initial “=” character, it will presume that some sort of constant is being entered into a cell, and compare against various patterns to determine what that is (e.g., YYYY-MM-DD would be a date, something matching a number format would be a number, etc.). Let us consider from here on the case where there is an initial “=”, to be interpreted as a formula.

If a user interface is intended to be similar to the exchange format, it will probably continue to use the semicolon as function separator, the exchange characters for operators, and so on. Thus, most of a simple user interface is usually easy to develop, simply by copying the syntax given this specification.

However, most user interfaces do not require markers to disambiguate most cell references, named expressions, and function names. Once these markers are removed from the exchange format, it may appear difficult to develop a user interface to distinguish between these cases, especially if the application supports extensions such as subtables (either cells and sheets inside cells) and function calls on named expressions. This is actually not as difficult as it may first appear; this section documents one way to do this.

First, notice that an Expression can be (among other things) a function call (beginning with a function name), a reference, or a named expression. Without markings or rules, some of their patterns would be ambiguous, especially since most user interfaces will treat function and column names in case-insensitive manner. Two things are needed: (1) rules to disambiguate, and (2) overriding patterns so that a human can easily force any particular interpretation. Ideally those rules would also handle phrases such as whole column ranges in a graceful manner.

Thus, the implementation can look for patterns, where higher-precedence matches would override lower-precedence ones. The following is a example of a list of such patterns, starting with the highest-precedence one, along with an example (in the UI and exchange format) and the meaning of the pattern; the {...} surround syntactic names:

Pattern	Example		Meaning
	UI	Exchange	
{Identifier} “(“	SUM(...	SUM(...	Function call to built-in function, such as SUM(...) or ERROR.TYPE(...). If there is no such function, search for a named expression of that name starting from the current sheet, and if found, use
	ERROR.TYPE(	ERROR.TYPE(...	

Copyright © OASIS Open 2006. All Rights Reserved.

	...		that (and consider it a call through the named expression). Use prefix $$ to force interpreting this as a named expression. Notice that function calls must have opening parentheses, disambiguating them from other possibilities.
{LetterXML} {LetterXML | DigitXML}* \. “$”? [A-Za-z]+ “$”? [0-9]+ ...	A1.B2	[A1.B2] or ['A1'.B2]	Sheet name and cell reference; “Sheet A1's cell B2”. Notice we prefer this interpretation to the less likely “cell A1, subtable cell B2”. This can't be a function call, because it didn't match the previous pattern. For arbitrary sheet names, surround the sheet name with '...'. Use prefix '...' followed by “.” to force this interpretation
“$”?[A-Za-z]+ “$”?[0-9]+	A1	[.A1]	Cell reference to current sheet; “cell A1 of current sheet”. Use prefix “.” to force this interpretation
“$”?[A-Za-z]+ “:” “$”? [A-Za-z]+	B:B	[.B:.B]	Full column range reference. Whitespace may surround the “:”.
{Identifier}	Hello	Hello or $ $Hello	If it isn't anything else, it must be a named expression. Use prefix $$ to force interpretation to be a named expression, which must always be used to call functions in named expressions (if that is permitted).

And here are forms that are completely unambiguous, allowing the user to force a particular interpretation:

Pattern	Example		Meaning
	UI	Exchange	
\ .[A-Za-z]+[0-9]+ ...	.A1	[.A1]	Unambiguous cell reference to current sheet; “cell A1 of current sheet”. The prefix is needed when referring to a subtable of a cell in the current sheet.
	.A1.B2	[.A1.B2]	
{QuotedSheetName} \ . [A-Za-z]+[0-9]+ ...	'A1'.B2	['A1'.B2]	Sheet name and cell reference, marked unambiguously, arbitrary sheet name.
' IRI ' # ...	'./myfile.ods'#Total	'./myfile.ods'#Total	Source IRIs are always unambiguous; they are surrounded by '...' followed by #.
“$$” ...	$$A1	$$A1 or $ $'A1'	Unambiguously marked named expression id, named A1.

This means you can have complex expressions like this:

UI Example	Meaning
$$A1.B2	Search for named expression name, “A1.B2” (a 5-character name), starting from current sheet
A1.$$B1	Go to sheet A1, then look in its named expression id "B1".
'http://oasis.org'#S1.M1	Get the document from oasis.org, then look in

Copyright © OASIS Open 2006. All Rights Reserved.

	sheet S1's cell M1
'http://fred'#'A1'.$$B2	External source fred, sheet A1, named expression id B2.
'http://fred'#B2	External source fred, (global) named expression B2.
$$Myfunc(1)	Call named expression Myfunc. Note that this is not in the current specification, but is a plausible extension.

Note again that this particular user interface is *not* mandated by this specification. It is simply an illustration to show how to create a user interface. Applications can innovate with new and better user interfaces, while still exchanging documents with others.

Copyright © OASIS Open 2006. All Rights Reserved.

6 Standard Operators and Functions

This section defines the predefined operators and functions. For the purpose of this specification, operators are simply functions that use a different syntax: they may be prefix, infix, or postfix operators, and that is noted in their definition. Implementations **may** (and typically do) define additional predefined functions, beyond what is defined in this specification, as long as those additions do not conflict with the requirements here. In addition, implementations **may** add additional optional parameters, or accept a wider domain of inputs, in these predefined functions, as long as they continue to meet the specification.

Implementations **may**, and typically do, display different function names than listed here in their user interface. For example, implementations **may** change the displayed function name depending on the current locale's language and/or a user interface "skin." An implementation **may** display a different function name because the implementation has traditionally used a different function name as well.

Function names ignore case, so "sum" and "SUM" refer to the same function. However, implementations **should** write function names in all uppercase letters when writing OpenFormula formulas.

Rationale: By writing function names in all upper case, the text is more likely to stay the same when a file is read and later rewritten, making textual comparison simpler. Also, function names are traditionally written in all uppercase for spreadsheet formulas. This may be due to the influence of the first spreadsheet, VisiCalc; VisiCalc first ran on the Apple][microcomputer, and early models of that microcomputer could not display lowercase letters in text mode unless it was modified.

Unless otherwise noted, if any value being provided is an error, the result is an error; if more than one error is provided, one of them is returned (applications **should** return the leftmost error result). Some "functions", such as PI() and TRUE(), always return the same constant value, and are considered constants.

TBD: Re-examine functions' definition for when they are in array formulas, to make sure they are adequately defined.

This section begins with a subsection describing the common template used to describe all functions and operators. This is followed by the subsection on implicit conversion operators (the operators that are implicitly called when a type is requested). The next subsection describes the standard operators such as "+" (add numbers) and "&" (concatenate text). This is followed by a number of different subsections defining functions; these subsections group similar functions together so that similar functions are near each other.

6.1 Common Template for Functions and Operators

For every function or operator, the following are defined in this specification:

- **Name:** The function/operator name, e.g., "SUM". The name is given in a specification header. Operators are named first with its location (Prefix, Infix, or Postfix), following by the word Operator and the actual name surrounded by double-quotes. Thus, Infix Operator "+" is easily distinguished from Prefix Operator "+".
- **Summary:** A one-sentence briefly describing the function or operator.
- **Syntax:** A prototype showing its syntax. In particular, it shows parameter names (in order), with each parameter prefixed by the type or pseudo-type required of that parameter. If the

Copyright © OASIS Open 2006. All Rights Reserved.

type has multiple names separated by “|”, then any of those types are permitted. Here { ... } indicates a list of zero or more parameters, separated by the function parameter separator character. A { ... } followed by a superscripted + indicates a list of one or more parameters, separated by the function parameter separator character. Components surrounded by [...] are optional, as is any parameter with the keyword optional. A parameter followed by the = symbol has the default value given after equal sign; any such parameter is always optional (even if it is not surrounded by [...]). Parameters are separated with a semicolon (";"), as per the OpenDocument format syntax; note that some implementations may display parameter separators with commas or some other alternative. Note that when a function is given a value of a different type, the parameters are first converted using the implicit conversion rules before the function operates on its parameters. Applications **may** extend functions by permitting fewer or additional parameters, which documents **may** use, but portable documents **shall not** use such implementation extensions.

- **Returns:** Return type (e.g., Number, Text, Logical, Reference).
- **Constraints:** A description of constraints, in addition to the constraints imposed by the parameter types. If there are no additional constraints beyond those imposed by the parameter types, this is "None". If a constraint is not met, the function/operator **should** return an Error unless otherwise noted.
- **Portable Constraints:** A description of constraints necessary for portability. Not all functions/operators include this. If these constraints are not met, results may differ between different implementations.
- **Semantics:** This text describes what the function/operator does. In some cases they will be defined by mathematical formulas or by an OpenFormula formula. Note that the mathematical formulas define the *correct mathematical result,* and *not* the algorithm for calculation. Since computing systems have limited precision and range of numbers, some functions *cannot* or *should not* be naively implemented as their mathematical equations suggest. It specifically *not* in the scope of this specification to describe and discuss these nuances, or to explain numerical algorithms. Instead, this specification simply defines the mathematically correct answer, and allows implementors to choose the best algorithm that will meet that definition. If a parameter is a pseudotype, but the provided value fails to meet the requirements for that type, a function/operator **may** return an Error value.
- **Test Cases:** A set of one or more test cases.
- **Comment:** Explanatory comment.
- **See also** A list of related operators and functions.

The implicit conversion operators omit many of these items, e.g., the syntax (since there is none).

6.2 Implicit Conversion Operators

Any given function or operand takes 0 or more parameters, and each of those parameters has an *expected type*. The expected type can be one of the base types, identified above. It can also be of some *conversion type* that controls conversion, e.g., *Any* means that no conversion is done (it can be of any type); *NumberSequence* causes a conversion to an ordered sequence of zero or more numbers. If the passed-in type does not match the expected type, an attempt is made to automatically convert the value to the expected type. An error is returned if the type cannot be converted (this can never happen if the expected type is Any). Unless otherwise noted, any conversion operation applied to a value of type Error returns the same value.

6.2.1 Conversion to Scalar

To convert to a scalar, if the value is of type:

Copyright © OASIS Open 2006. All Rights Reserved.

- Number, Logical, or Text, return the value.
- reference to a single cell: obtain the value of the referenced cell, and return that value.
- reference to more than one cell: do an implied intersection (described below) to determine which single cell to use, then handle as a reference to a single cell.

6.2.4 Conversion to Number

If the expected type is Number, then if value is of type:

- Number, return it.
- Logical, return 0 if FALSE, 1 if TRUE.
- Text: The specific conversion is implementation-defined; an application may return 0, an error value, or the results of its attempt to convert the text value to a number (and fall back to 0 or error if it fails to do so). Applications may apply VALUE() or some other function to do this conversion, should they choose to do so. Conversion depends on the actual locale the application runs in, especially if group or decimal separators are involved. Note that portable spreadsheet files cannot depend on any particular conversion, and **shall** avoid implicit conversions from text to number.

  Note: Many implementors recommend to generate an error for an attempted conversion instead.
- Reference: If the reference covers more than one cell, do an implied intersection to determine which cell to use. Then obtain the value of the single cell and perform the rules as above. If the calculation setting “precision-as-shown” is true, then convert the number to the closest possible representation of the displayed number. If the cell is empty (blank), use 0 (zero) as the value. Applications **may** choose to convert references to text in a different manner than they handle converting embedded text to a number.

Note: Semantics vary in current applications:

- Excel, Gnumeric, and SheetToGo use "Excel" semantics: If they encounter Text or a reference to Text, they always convert the text value to a number (using VALUE()) and return the number or error. (Walkenbach, 2004) Under these semantics, COS("hi") first computes VALUE("hi"), producing an Error, and COS() applied to an error value produces an error value.
- Lotus 1-2-3v9, Quattro Pro, KSpread use "Lotus" semantics: If they encounter Text or a reference to Text, they always return the number 0. Under these semantics, COS("hi") computes COS(0), producing 1.
- OpenOffice.org 2 splits the difference: Inline text is converted to a number (like Excel), but references to text are always considered 0 (even if they could be converted to a different number, and *would* be converted to a different number if in-line). Thus, in OOo 2, if B3 has the string value "7", B3+1 is 1, but "7"+1 and (B3&"")+1 are both 8.

Expression	***Result***	***Comment***
=(1=1)+2	3	Inline logical True is converted to 1.
=[.B5]+[.B6]	4	Adding forces conversion of TRUE to 1, even if by reference
=IF(ISERROR(“7”+0);TRUE();OR(("7"+0)=7; ("7"+0)=0))	True	Since “+” forces a conversion to number, passing "7" to “+” should force a conversion to 0, 7, or an error.

Copyright © OASIS Open 2006. All Rights Reserved.

=IF(ISERROR([.B3]+0); TRUE();OR(([.B3]+0)=7; ([.B3]+0)=0))	True	B3 is "7"; it should convert to 0, 7, or an error. Note that [.B3]+0 may produce a different result than inline "7"+0.
=ISERR(IF(COS("hi")=1; 1/0;0))	True	Functions expecting a number but get non-numeric text convert the number to 0 or an Error.
=5+[.B8]	5	Empty cells in a number context are automatically converted into 0.

6.2.5 Conversion to Integer

If the expected type is Integer for a function or operator, apply the “Conversion to Number” operation. Then, if the result is a Number but not an integer, apply the specific conversion from Number to integer specified by that particular function/operator. If the function or operator does not specify any particular conversion operation, then the conversion from a non-integer Number into an integer is implementation-defined.

Many different conversions from a non-integer number into an integer are possible. The conversion direction may be towards negative infinity, towards positive infinity, towards zero, away from zero, towards the nearest even number, or towards the nearest odd number. A conversion can select the nearest integer, the nearest even or odd integer, or simply the “next” integer in the given direction if it is not already an integer. If a conversion selects the nearest integer, a direction is still needed (for when a value is halfway between two integers). In this specification, this conversion is referred to as “rounding” or “truncation”; these terms by themselves do not specify any specific operation.

If a function specifies its rounding operation using a series of capital letters, the function defined in this specification for that function is used to do the conversion to integer. Common such functions are:

- INT, which if given non-integer rounds down to the next integer towards negative infinity, regardless of whether or not it is the closest integer.
- ROUND, which if given non-integer rounds to the nearest integer. If the input number is halfway between integers, it rounds away from zero.
- TRUNC, which if given non-integer rounds towards zero, regardless of whether or not that integer is the closest integer.

Rationale: It'd be nice if we could simply say, “all functions expecting an integer first use INT on their input parameters,” or some other function. But this is not true in practice, and many users have built important documents that depend on the differing rounding properties of different functions. E.G., many functions use INT, but ISEVEN uses TRUNC. It would be extremely risky to users if the conversion to integer function silently changed on them, because it could cause documents to silently produce the wrong answers. Users who want an conversion that is different from what the function does, or using one with implementation-defined behavior, can use the rounding functions already available.

TODO: Walk through functions, and change all “Number” in input or result types to Integer where appropriate. Document which integer conversion function is used (usually INT), and include test cases to check that.

Copyright © OASIS Open 2006. All Rights Reserved.

6.2.9 Conversion to Logical

If the expected type is Logical, then if value is of type:

- Number, return TRUE() for nonzero and FALSE() for 0.
- Text: The specific conversion is implementation-defined; an application may return False, an error value, or the results of its attempt to convert the text value (ignoring case) to a logical value (and fall back to False or error if it fails to do so). Conversion depends on the actual locale the application runs in. Note that portable spreadsheet files cannot depend on any particular conversion, and **shall** avoid implicit conversions from text to logical.

  Note: A typical algorithm for converting a text value to a logical value is to determine if the text matches the text "TRUE" ignoring case, then return TRUE, if the text matches the text "FALSE" ignoring case, then return FALSE, otherwise return Error. Typically case is ignored no matter what the value of `table:case-sensitive` is. It is implementation-defined what happens if the text does not match "TRUE" or "FALSE" in a case-sensitive manner- it may return a logical value or an Error.

  Note: Many Implementors recommend to generate an error for an attempted conversion instead.
- Logical, return it.
- Reference, convert to scalar and then perform as above. If the reference is to an empty cell, consider it FALSE().

Expression	*Result*	*Comment*
=IF(5;TRUE();FALSE())	True	Nonzero considered True.
=IF(0;TRUE();FALSE())	False	Zero considered False.

Note: Semantics vary in current applications:

- In Excel, Gnumeric and SheetToGo, the text "True" and "False" (ignoring case) are automatically converted to True and False, else an error.
- On others, such as Lotus 1-2-3 Quattro Pro, and OpenOffice.org 2, this is handled like text to number. So on Lotus 1-2-3 and Quattro Pro, text is always considered 0, so all text is considered False.
- On OpenOffice.org, inline text is converted following Excel-like rules, but referenced text is always considered 0 and thus always False.

Additionally, conversion may differ for specific functions explicitly expecting values of type Logical, such as AND and OR, where textual cell content may get ignored completely. The following table illustrates the current behavior of some applications in an English language locale, assuming a cell content of:
A1: "false" A2: "true" A3: "blah"

Expression	OOo	Gnumeric	Kspread	Excel
=IF(A1;TRUE();FALSE())	FALSE	FALSE		FALSE
=IF(A2;TRUE();FALSE())	FALSE	TRUE		TRUE
=IF(A3;TRUE();FALSE())	FALSE	Error		Error

Copyright © OASIS Open 2006. All Rights Reserved.

=IF("false";TRUE();FALSE())	FALSE	FALSE		FALSE
=IF("true";TRUE();FALSE())	TRUE	TRUE		TRUE
=IF("blah";TRUE();FALSE())	Error	Error		Error
=AND(A1)	Error	TRUE		Error
=AND(A2)	Error	TRUE		Error
=AND(A3)	Error	TRUE		Error
=AND("false")	Error	TRUE		FALSE
=AND("true")	Error	TRUE		TRUE
=AND("blah")	Error	TRUE		Error
=AND(A1;TRUE())	TRUE	TRUE		TRUE
=AND(A2;TRUE())	TRUE	TRUE		TRUE
=AND(A3;TRUE())	TRUE	TRUE		TRUE
=AND("false";TRUE())	Error	TRUE		FALSE
=AND("true";TRUE())	Error	TRUE		TRUE
=AND("blah";TRUE())	Error	TRUE		Error

TODO: add Kspread 1.6 behavior, which is different from 1.4; said to be similar to OOo with the exception that inline text and and textual cell content is treated identical. Note that this is merely a TODO for the annotations, not for the specification.

6.3 Standard Operators

The standard operators are simply functions with special syntax. The binary operations +, -, *, /, and ^, when provided two numbers, compute (respectively) the addition, subtraction, multiplication, division, and exponentiation of those numbers. The unary operation "-" produces the negative of this number. Unary + simply passes on its operand's value, unchanged, with exactly the same type. The string concatenation operator "&" converts any non-string values into strings before concatenating them.

6.3.1 Infix Operator "+"

Summary: Add two numbers.

Syntax: *Number Left* + *Number Right*

Returns: Number

Constraints: None

Copyright © OASIS Open 2006. All Rights Reserved.

Semantics: Adds numbers together. Due to the way conversion works, logical values are converted to numbers.

Test Cases:

Expression	*Result*	*Comment*
=1+2	3	Simple addition.
=[.B4]+[.B5]	5	2+3 is 5.

See also Infix Operator "-", Prefix Operator "+"

6.3.2 Infix Operator "-"

Summary: Subtract the second number from the first.

Syntax: *Number Left - Number Right*

Returns: Number

Constraints: None

Semantics: Due to the way conversion works, logical values are converted to numbers.

Test Cases:

Expression	*Result*	*Comment*
=3-1	2	Simple subtraction.
=[.B5]-[.B4]	1	3-2 is 1.
=5 - - 2	7	Subtraction can be combined with unary minus.
=5--2	7	Subtraction can be combined with unary minus, even without spaces
=3-2+3	4	Left-to-right associative
=[.C8]-[.C7]	365	Difference of two dates is the number of days between them.

See also Infix Operator "+", Prefix Operator "-"

6.3.3 Infix Operator "*"

Summary: Multiply two numbers.

Syntax: *Number Left * Number Right*

Returns: Number

Constraints: None

Semantics: Multiplies numbers together. Due to the way conversion works, logical values are converted to numbers.

Copyright © OASIS Open 2006. All Rights Reserved.

Test Cases:

Expression	*Result*	*Comment*
=3*4	12	Simple multiplication.
=[.B4]*[.B5]	6	2*3 is 6.
=2+3*4	14	Multiplication has a higher precedence than addition.

See also Infix Operator "+", Infix Operator "/"

Note: Excel 2002 cannot use "*" to multiply two complex numbers together; instead, you have to use IMPRODUCT. This is, of course, very inconvenient, but many other applications do the same thing.

6.3.4 Infix Operator "/"

Summary: Divide the second number into the first.

Syntax: *Number Left* / *Number Right*

Returns: Number

Constraints: None

Semantics: Divides numbers. Due to the way conversion works, logical values are converted to numbers. Dividing by zero returns an Error. Applications **shall** support fractions, so 1 / 2 must produce 0.5, not 0.

Test Cases:

Expression	*Result*	*Comment*
=6/3	2	Simple division.
=144/3/2	24	Division is left-to-right associative
=6/3*2	4	Division and multiplication are left-to-right
=2+6/2	5	Division has a higher precedence than +.
=5/2	2.5	Simple division; fractional values are possible.
=1/0	Error	Dividing by zero is not allowed.

See also Infix Operator "-", Infix Operator "*"

6.3.5 Infix Operator "^"

Summary: Exponentiation (Power).

Syntax: *Number Left* ^ *Number Right*

Returns: Number

Copyright © OASIS Open 2006. All Rights Reserved.

Constraints: OR(Left != 0; Right != 0)

Portable Condtraints: NOT(AND(Left=0; Right=0))

Semantics: Returns POWER(Left, Right). Due to the way conversion works, logical values are converted to numbers. Implementations **shall** compute 0^0 as one of 0, 1, or an Error, but it is implementation-defined which of these options is the result.

Note: In Excel, 0^0 is Error (more specifically, #NUM!). Future versions of this specification may be more specific. OpenOffice.org 2.4, as well as most mathematical texts, treat 0^0 as 1.

Test Cases:

Expression	*Result*	*Comment*
=2^3	8	Simple exponentiation.
=9^0.5	3	Raising to the 0.5 power is the same as a square root.
=(-5)^3	-125	Must be able to accept Left < 0.
=4^-1	0.25	Must be able to accept Right < 0.
=5^0	1	Raising nonzero to the zeroth power results in 1.
=0^5	0	Raising zero to nonzero power results in 0.
=2+3*4^2	50	Precedence: ^ is higher than *, which is higher than +.
=-2^2	4	Unary "-" has a higher precedence than "^".
=(2^3)^2	64	8^2 is the square of 8
=2^(3^2)	512	2 to the ninth power
=2^3^2	64	“^” is left-associative, not right-associative

Rationale: This format has both ^ and POWER(), even though they are semantically the same thing. Most spreadsheet implementations supply both, allowing users to use whichever representation they prefer. Since this specification also supports both, it supports roundtripping the user's preferred representation. Some users intentionally use ^ in some contexts, and POWER() in others; without both representations it would be difficult to reconstruct a user's preferences.

There has been a lot of discussion about the precedence of binary "^" compared to unary "-". It was felt that we (the formula subcommittee) could not be silent about precedence. The precedence of prefix “-” being higher than infix “^” causes the expression “-2^2” to compute as 4, and is shared by Excel, OpenOffice.org 2, and Gnumeric. This is not universal; prefix "-" has a lower precedence on Lotus 1-2-3, Quattro Pro, SheetToGo, and Excel's own Visual Basic (Walkenbach, 2004, pg. 579), so these products will need to insert and remove parentheses when reading/writing expressions in OpenFormula where this matters. Mandating anything else would greatly increase confusion and create an unnecessary difference with actual practice in many spreadsheets still in use. The impact of this is expected to be rare; many documents with formulas do not use ^ at all, and those who are genuinely concerned about precedence are likely to parenthesize anyway. OpenFormula can also be used for attribute `draw:formula` as defined in OpenDocument 1.0 section 9.5.5 and attribute `anim:formula` as defined in 13.3.2; neither of these attributes includes an exponentiation operator "^", so this issue of precedence causes no problem. (Note that they require "," as the function separator, and have other requirements, as discussed later in this document).

Copyright © OASIS Open 2006. All Rights Reserved.

Note that “^” is *left*-associative, not *right*-associative, because many spreadsheet applcations do this: Excel 2000, OpenOffice.org 2.0, Lotus 1-2-3v9.8, SheetToGo, and Corel Quattro Pro 12. This is not universal; Gnumeric 1.4.3 is right-associative. Kspread and wikiCalc were originally right-associative, but after discussing associativity in the formula subcommittee, they agreed to switch to left-associativity. Having it be left-associative simplifies implementation; implementations can treat all infix operators as left-associative.

See also Infix Operator "*", POWER

6.3.6 Infix Operator "="

Summary: Report if two values are equal

Syntax: *Scalar* Left = *Scalar* Right

Returns: Logical

Constraints: None

Semantics: Returns TRUE if two values are equal. If the values differ in type, return FALSE. If the values are both Number, return TRUE if they are considered equal, else return FALSE. If they are both Text, return TRUE if the two values match, else return FALSE. For Text values, if the calculation setting `table:case-sensitive` is `false`, text is compared but characters differencing only in case are considered equal. If they are both Logicals, return TRUE if they are identical, else return FALSE. Note that, like most other functions, errors are propagated; error values *cannot* be compared to a constant error value to determine if that is the same error value.

Note that the result of “1=TRUE()” is implementation-defined. On implementations where logicals are implemented as numbers, this is true. However, on implementations where logicals are a separate distinct type, this would be normally be false.

Note that in most implementations, numbers are computed using fixed-length representations in base 2 or 16, using a matissa and an exponent. This means that values such as 0.1 cannot be exactly represented (just as 1/3 cannot be exactly represented in base 10 using decimal notation). As a result, (0.1*10) will not have the same bit representation as 1. Since many spreadsheet users do not understand how computers typically represent numbers, applications **may** attempt to hide these differences by allowing “nearly” equal numbers to be considered equal, but applications are not required to do so.

Note that in some user interfaces this is displayed or accepted as “==”.

Test Cases:

Expression	***Result***	***Comment***
=1=1	True	Trivial comparison.
=[.B4]=2	True	References are converted into numbers, and then compared.
=1=0	False	Trivial comparison.
=3=3.0001	False	Grossly wrong equality results are not acceptable. Spreadsheets cannot "pass" automated tests by simply making "=" always return TRUE when it's even slightly close.
="Hi"="Bye"	False	Trivial text comparison - no match.
=FALSE()=FALSE()	True	Can compare logical values.
=TRUE()=FALSE()	False	Can compare logical values.

Copyright © OASIS Open 2006. All Rights Reserved.

="5"=5	False	Different types are not equal.
=ISNA(NA()=NA())	True	If there's an error on either side, the result is an error -- even if you're comparing the "same" error on both sides.
=“Hi”=”HI”	True	Note, this is only true if `table:case-sensitive` is `false`.

See also Infix Operator "<>"

Note: OpenOffice.org's equal operator performs case sensitive or case-insensitive matching, depending on the setting of Tools.Options.Calc.Calculate "Case sensitive". By default, OOo2 is case-sensitive, while most others are case-insensitive. In an OpenDocument file the setting is stored in the `table:case-sensitive` attribute.

Note: It'd be nice to devise a more specific definition of “=” for “nearly equal numbers”, but after discussion we were unable to agree on one.

6.3.7 Infix Operator "<>"

Summary: Report if two values are not equal

Syntax: *Any* Left <> *Any* Right

Returns: Logical

Constraints: None

Semantics: Returns NOT(Left = Right) if Left and Right are not Error. Note that for Text values, if the calculation setting `table:case-sensitive` is `false`, text is compared but characters differencing only in case are considered equal.

Note that if either Left and Right are an error, the result is an error; this operator cannot be used to determine if two errors are the same kind of error.

Note that in some user interfaces, this is displayed (or accepted) as “!=” or “≠”.

Test Cases:

Expression	***Result***	***Comment***
=1<>1	False	1 really is 1.
=1<>2	True	1 is not 2.
=1<>"1"	True	Text and Number have different types.
="Hi"<>"HI"	False	Text comparison ignores case distinctions. Note, this is only true if `table:case-sensitive` is `false`.
=1/0=1/0	Error	This operator cannot compare error values to determine if they are the same error. If either side is an error, the result is an error.

See also Infix Operator "="

Copyright © OASIS Open 2006. All Rights Reserved.

Note: We don't need many test cases here, because it's defined in terms of another function that is tested.

6.3.8 Infix Operator Ordered Comparison ("<", "<=", ">", ">=")

Summary: Report if two values have the given order

Syntax: *Scalar* Left *op* *Scalar* Right

where *op* is one of: "<", "<=", ">", or ">="

Returns: Logical

Constraints: None

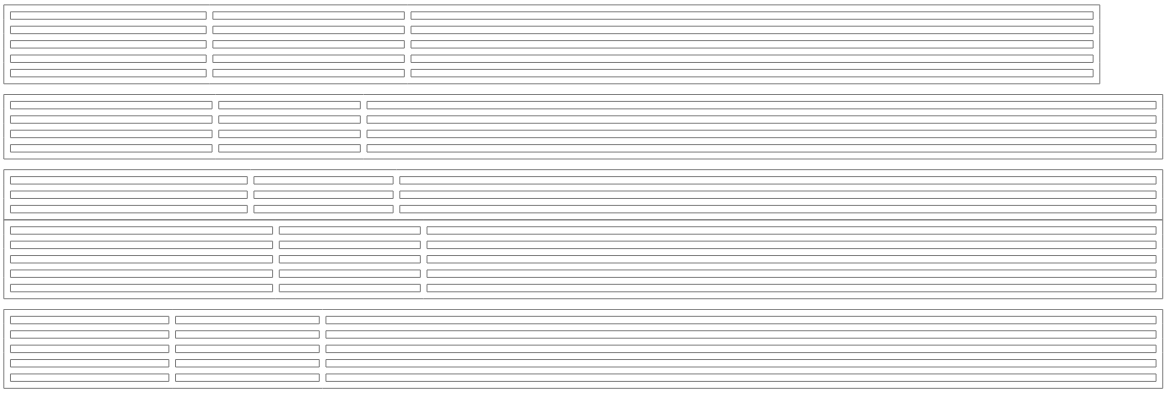
Semantics: Returns TRUE if the two values are less than, less than or equal, greater than, or greater than or equal (respectively). If both Left and Right are Numbers, compare them as numbers. If both Left and Right are Text, compare them as text; if the calculation setting `table:case-sensitive` is `false`, text is compared but characters are compared ignoring case. If the values are both Logical, convert both to Number and then compare as Number.

These functions return one of True, False, or an Error if Left and Right have different types, but it is implementation-defined which of these results will be returned when the types differ.

Test Cases:

Expression	***Result***	***Comment***
=5<6	True	Trivial comparison.
=5<=6	True	Trivial comparison.
=5>6	False	Trivial comparison.
=5>=6	False	Trivial comparison.
="A"<"B"	True	Trivial comparison of text.
="a"<"B"	True	True when `table:case-sensitive` is `false`.
="AA">"A"	True	Longer text is "larger" than shorter text, if they match in case-insensitive way through to the end of the shorter text.

See also Infix Operator "<>", Infix Operator "="



Copyright © OASIS Open 2006. All Rights Reserved.

6.3.14 Prefix Operator "+"

Summary: No operation; simply returns its one argument.

Syntax: + *Any Right*

Returns: Any

Constraints: None

Semantics: Returns the value given to it. Note that this does **not** convert a value to the Number type. In fact, it does *no* conversion at all of a Number, Logical, or Text value - it returns the same Number, Logical, or Text value (respectively). The "+" applied to a reference may return the reference, or an Error.

Test Cases:

Expression	*Result*	*Comment*
=+5	5	Numbers don't change
=+"Hello"	"Hello"	Does not convert a string to a number.

See also Infix Operator "+"

Note: In OOo2, SUM(+[.B4:.B5]) produces 5, because the "+" operator causes no change and just returns the range reference. In Excel 2003, it produces a value error; prefix "+" is not permitted on references.

6.3.15 Prefix Operator "-"

Summary: Negate its one argument.

Syntax: - *Number Right*

Returns: Number

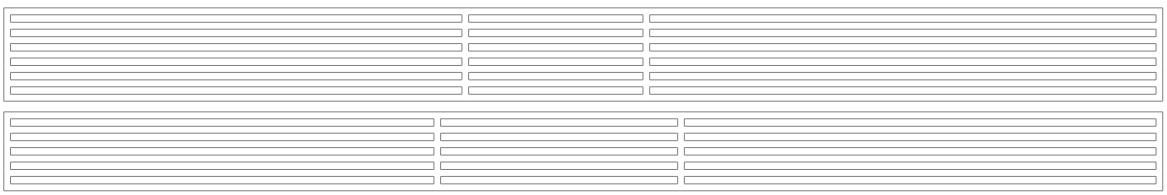
Constraints: None

Semantics: Computes 0 - Right.

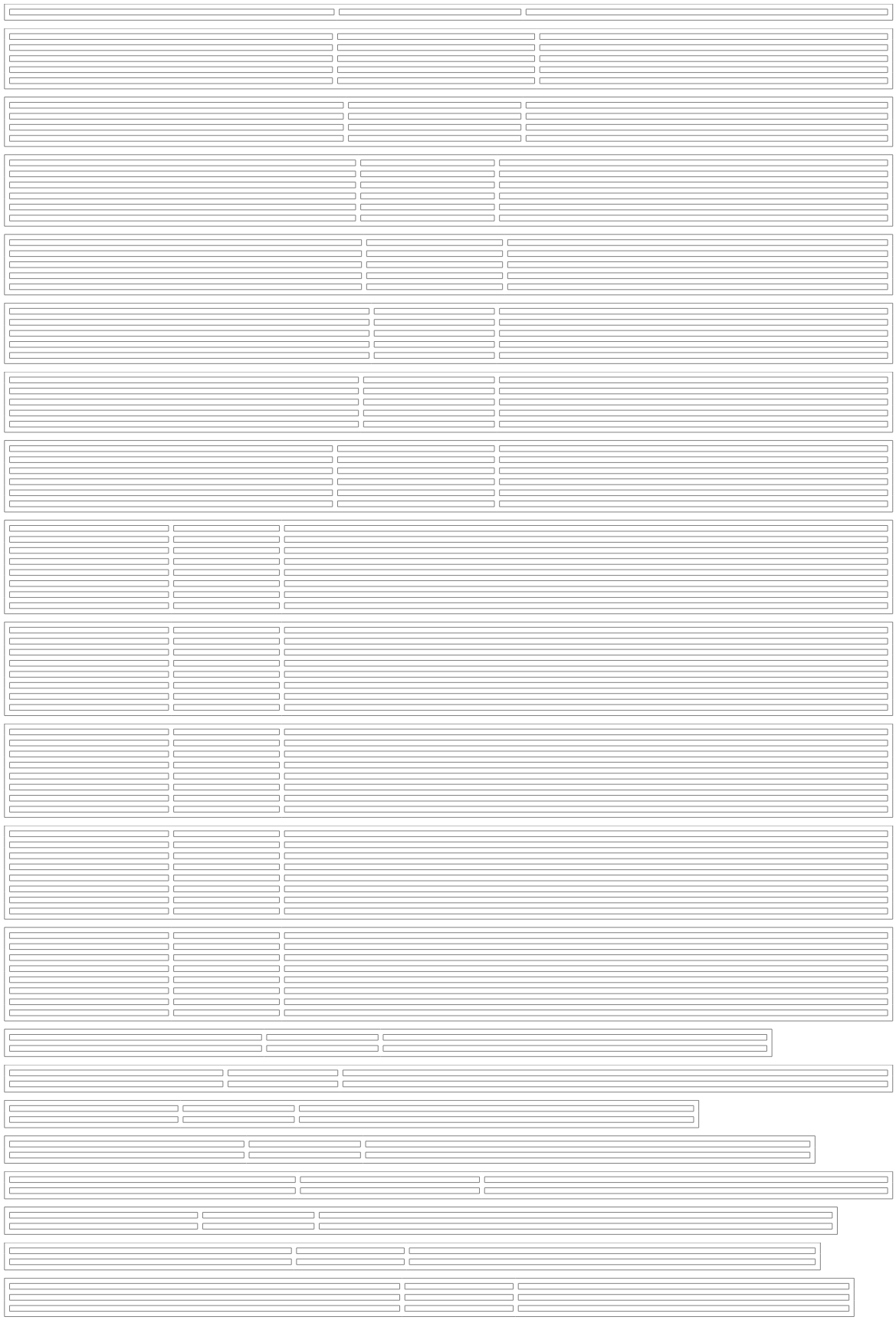
Test Cases:

Expression	*Result*	*Comment*
=-[.B4]	-2	Negated 2 is -2.
=-2=(0-2)	True	Negative numbers are fine.

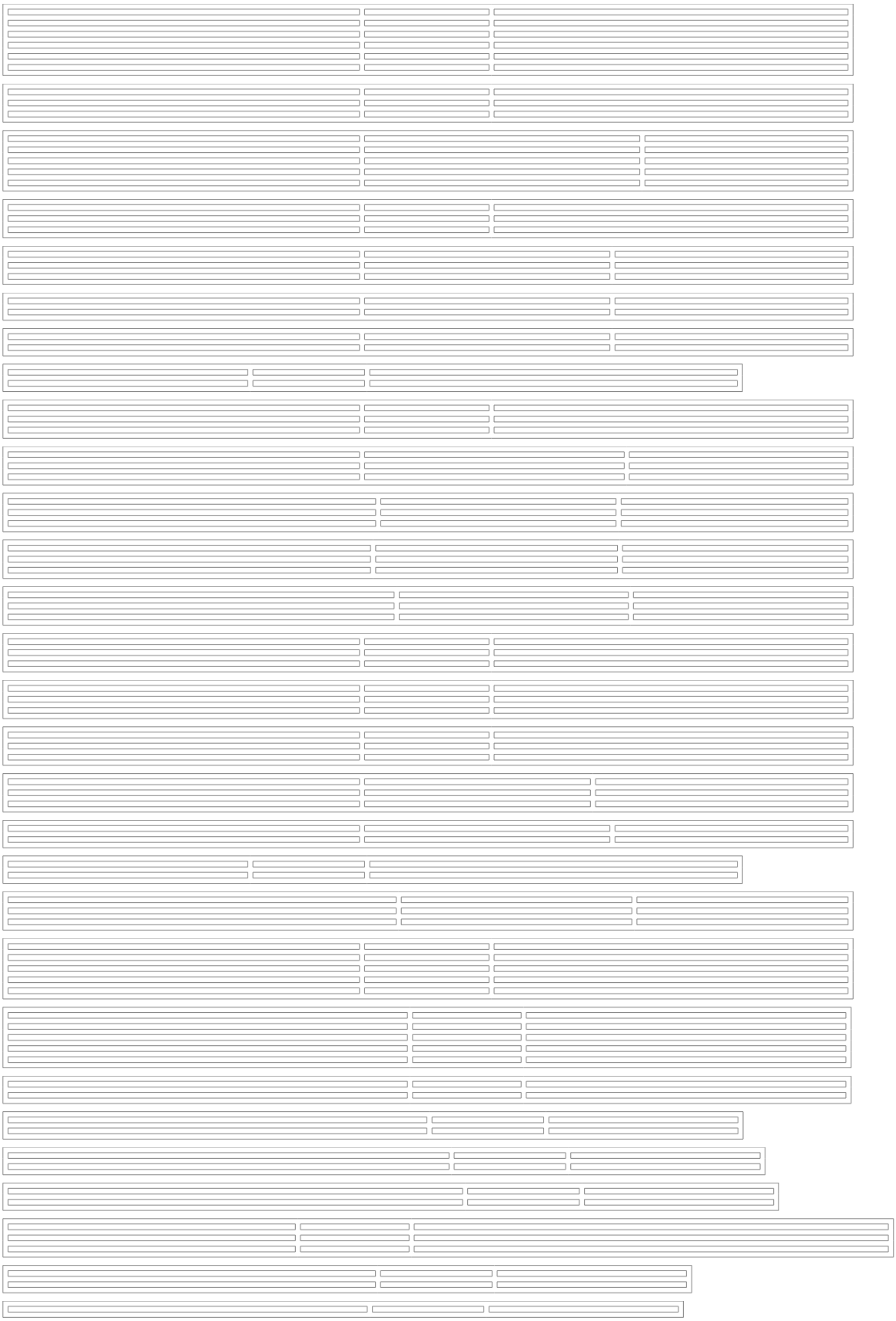
See also Infix Operator "-"



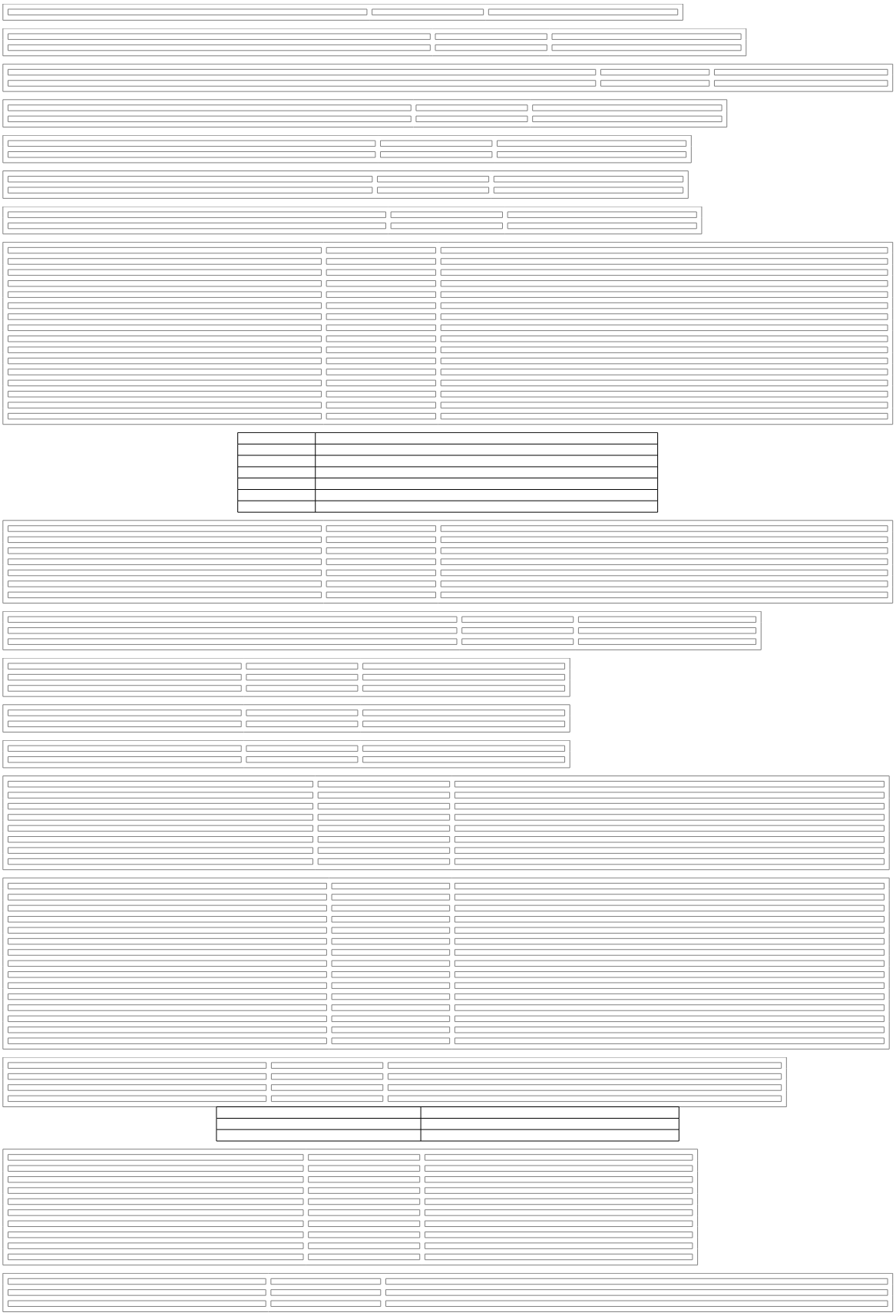
Copyright © OASIS Open 2006. All Rights Reserved.



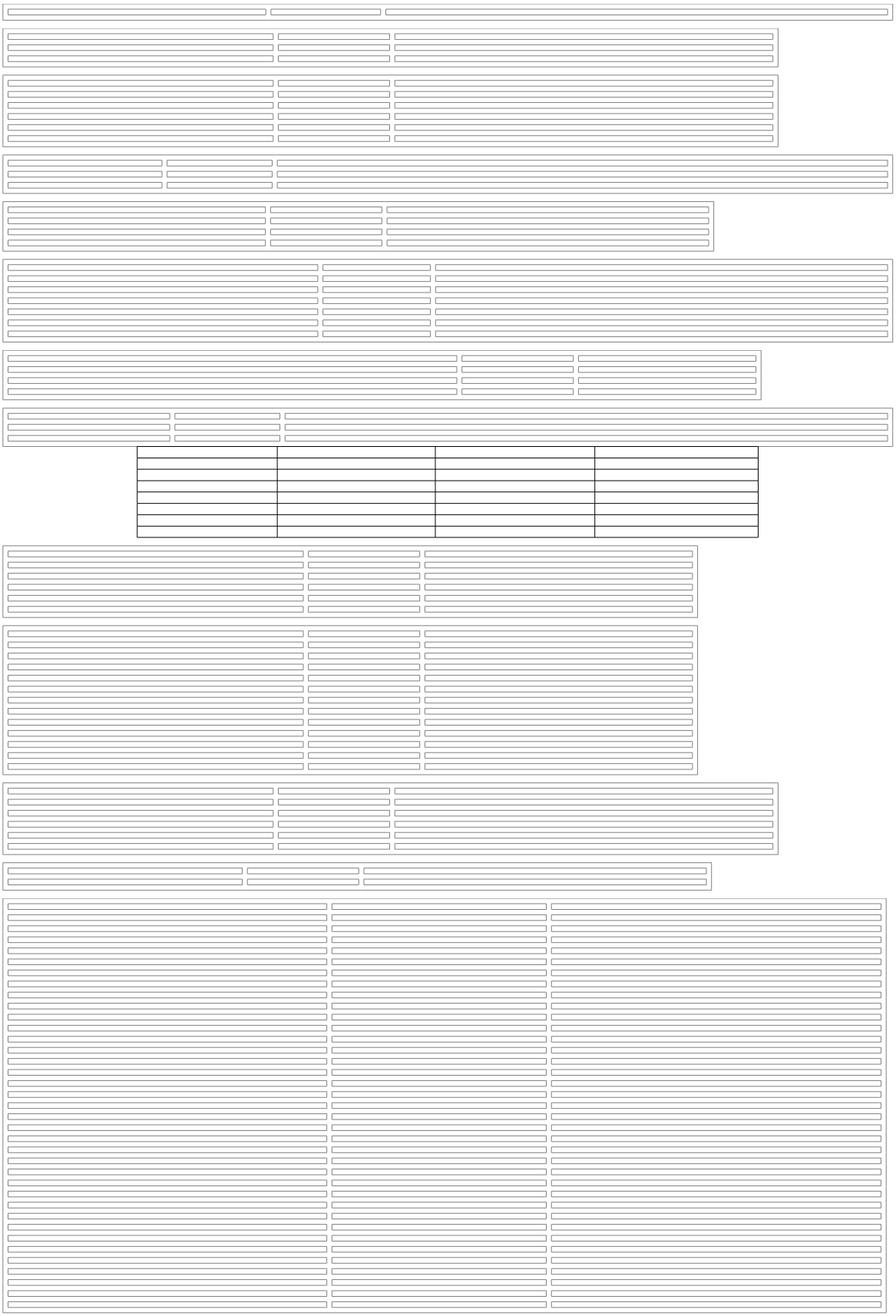
Copyright © OASIS Open 2006. All Rights Reserved.



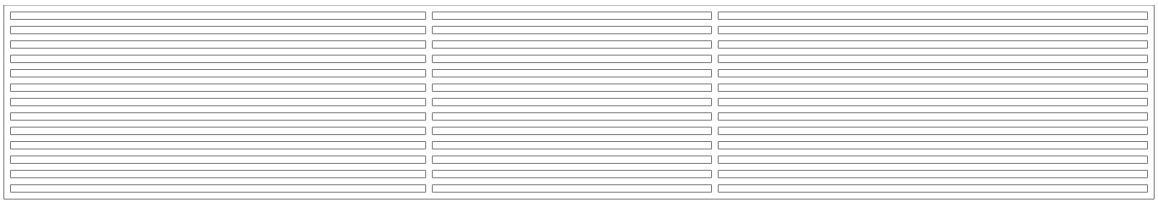
Copyright © OASIS Open 2006. All Rights Reserved.

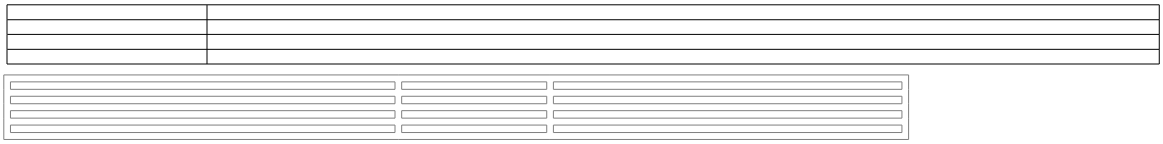


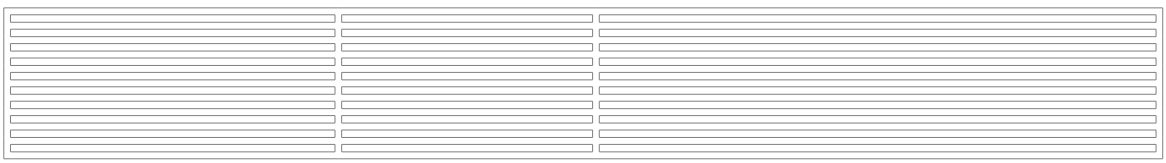
Copyright © OASIS Open 2006. All Rights Reserved.

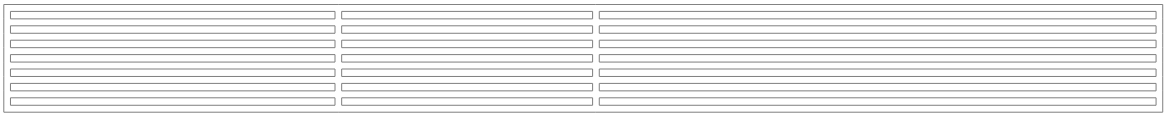


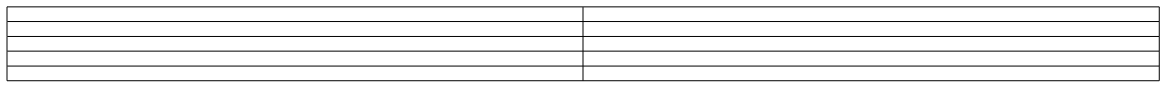
Copyright © OASIS Open 2006. All Rights Reserved.

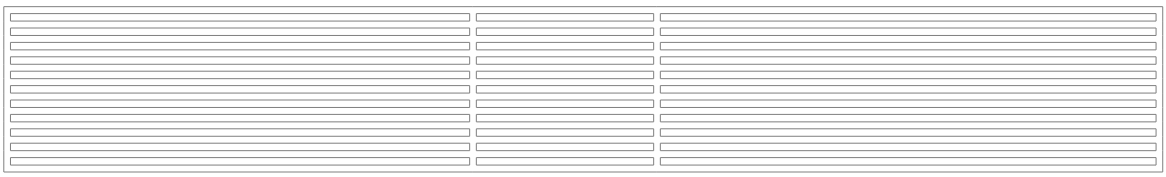


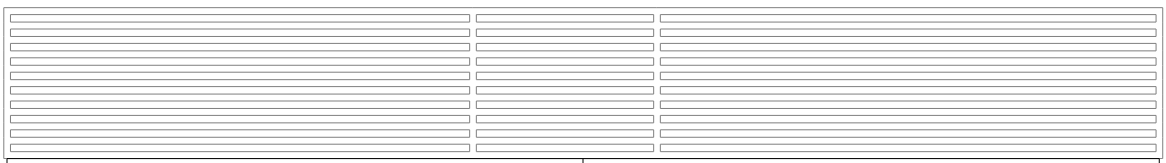


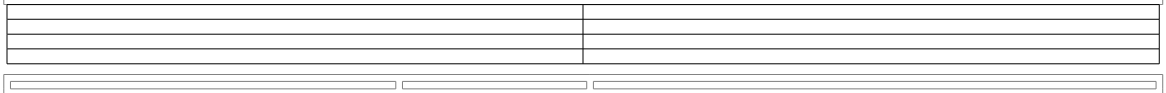


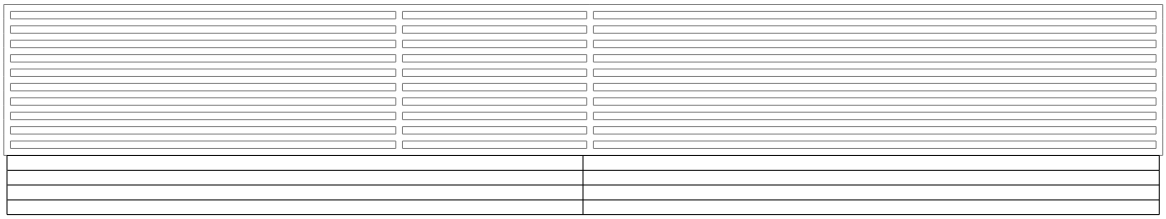




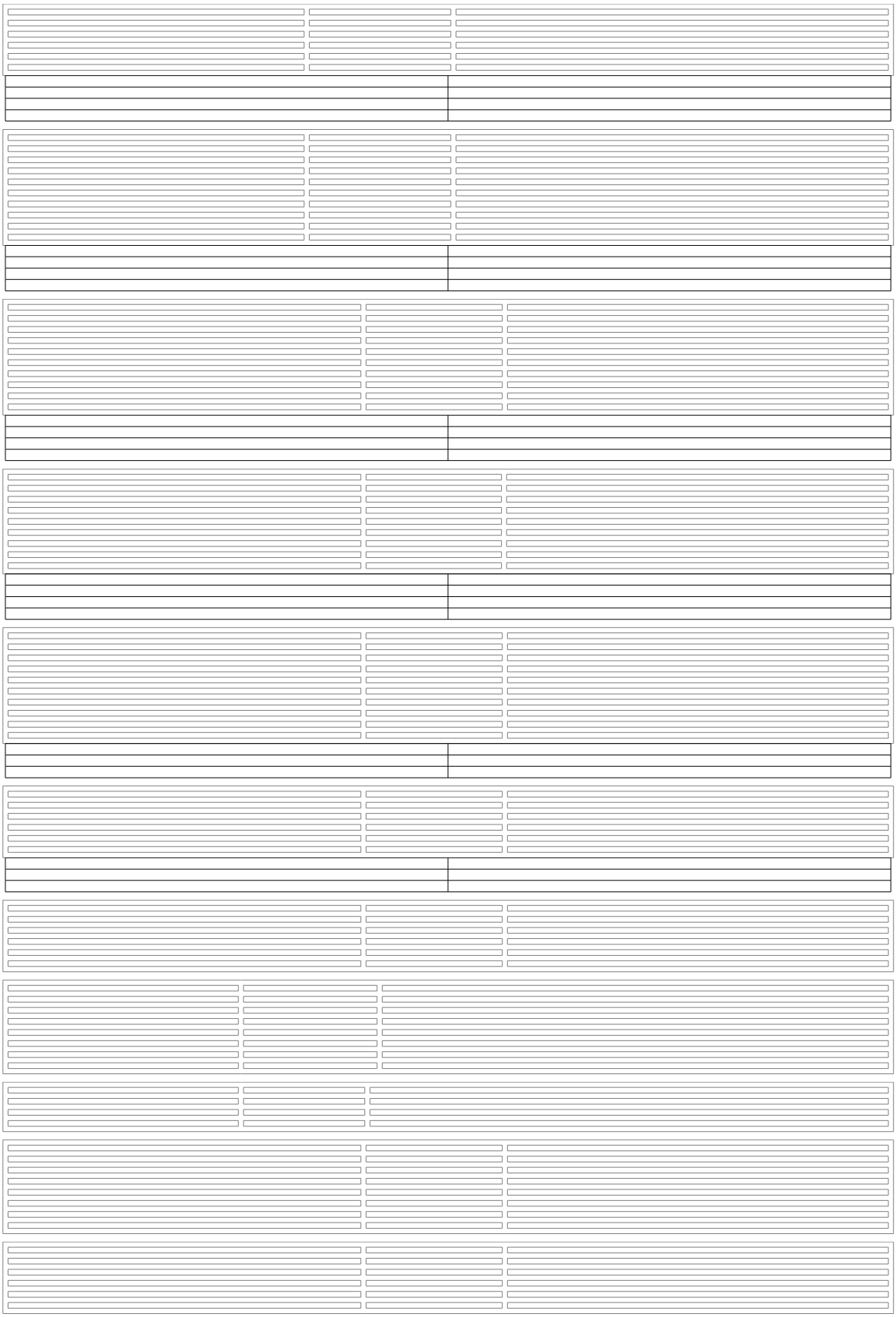




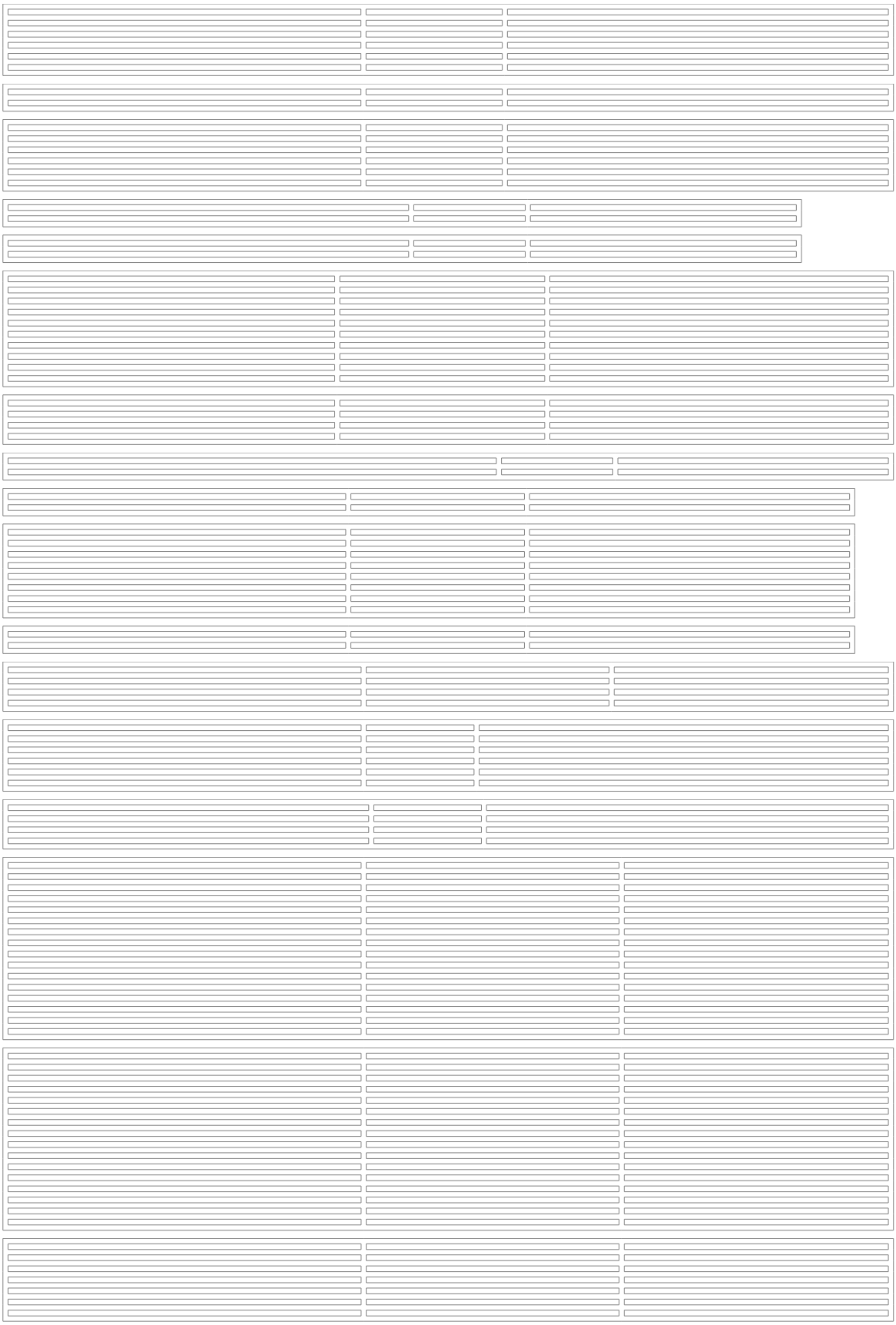




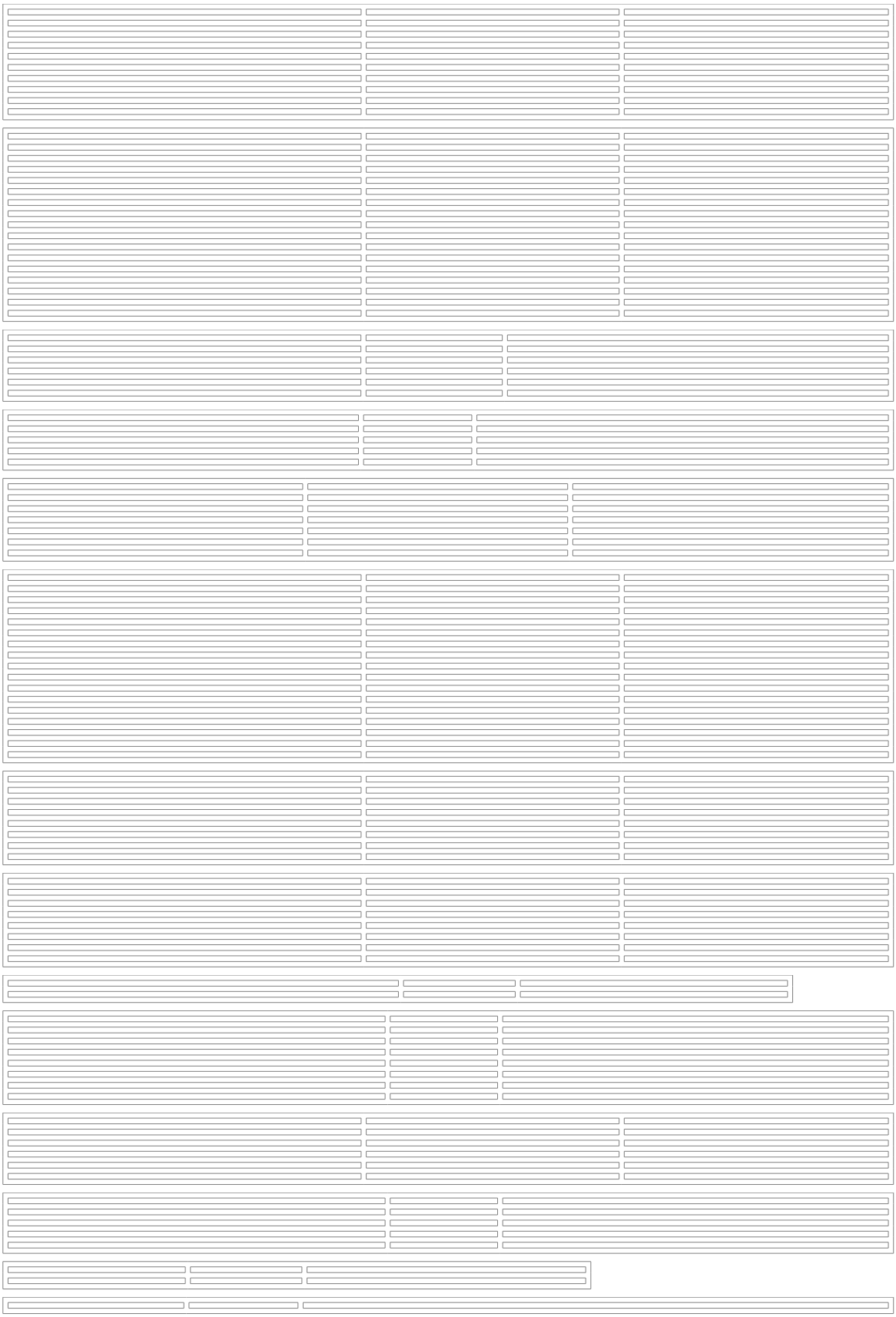
Copyright © OASIS Open 2006. All Rights Reserved.



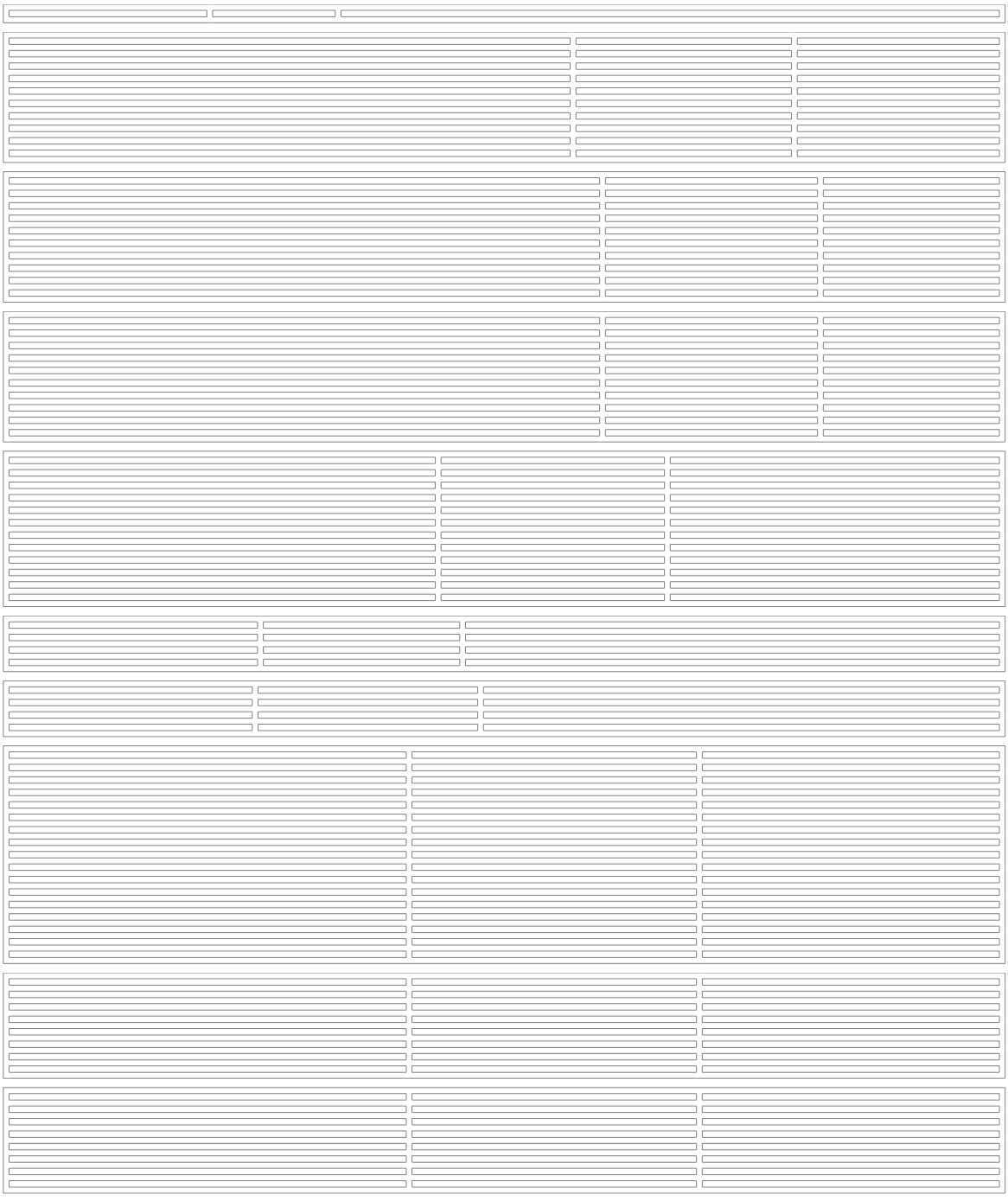
Copyright © OASIS Open 2006. All Rights Reserved.



Copyright © OASIS Open 2006. All Rights Reserved.



Copyright © OASIS Open 2006. All Rights Reserved.



6.12 Information Functions

Information functions provide information about a data value, the spreadsheet, or underlying environment, including special functions for converting between data types.

Copyright © OASIS Open 2006. All Rights Reserved.

6.12.13 ISBLANK

#TODO JH: necessary?#

Summary: Return TRUE if the referenced cell is blank, else return FALSE

Syntax: ISBLANK(*Scalar* X)

Returns: Logical

Constraints: None

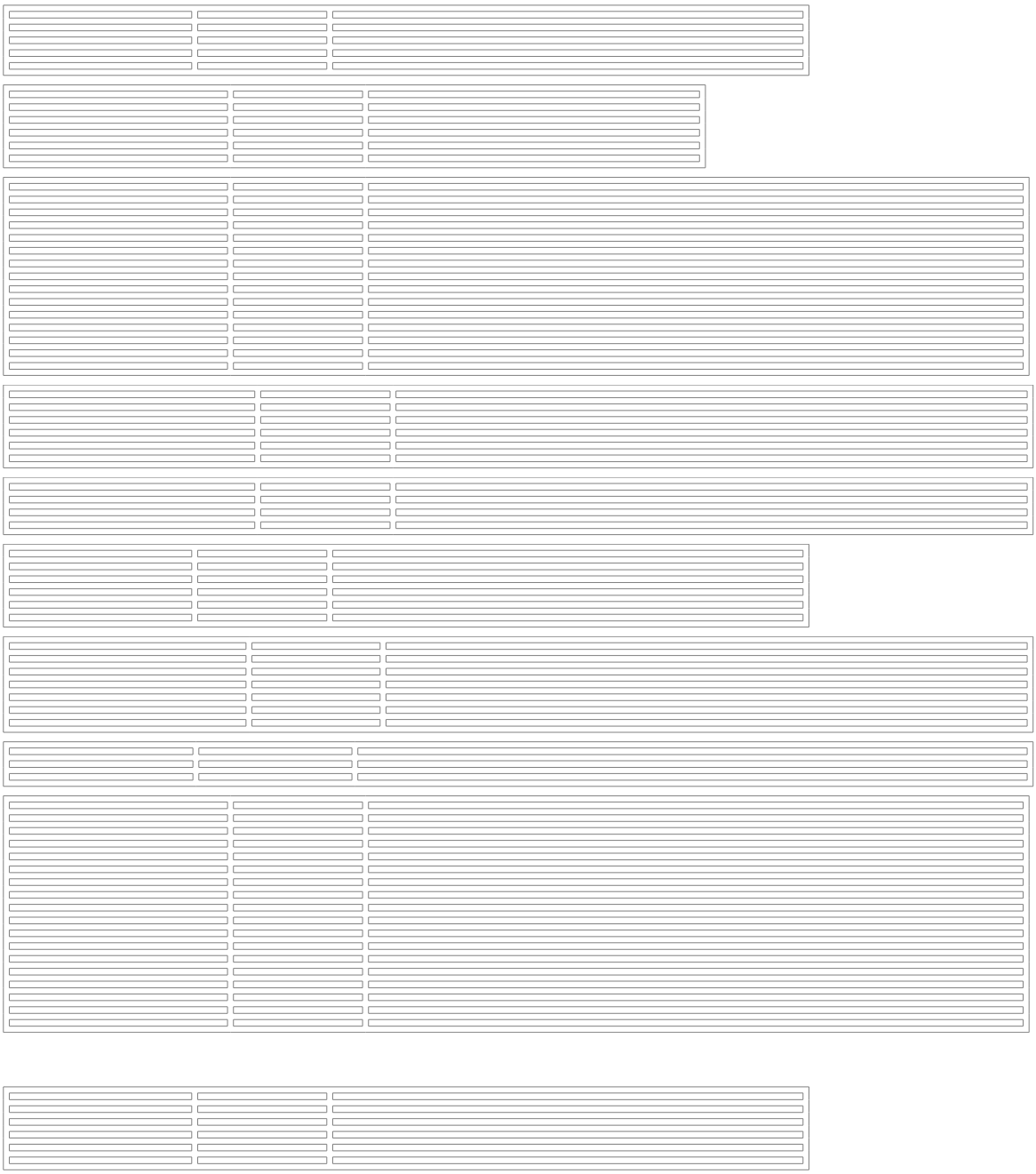
Semantics: If X is of type Number, Text, or Logical, return FALSE. If X is a reference to a cell, examine the cell; if it is blank (has no value), return TRUE, but if it has a value, return FALSE. A cell with the empty string is *not* considered blank.

Test Cases:

Copyright © OASIS Open 2006. All Rights Reserved.

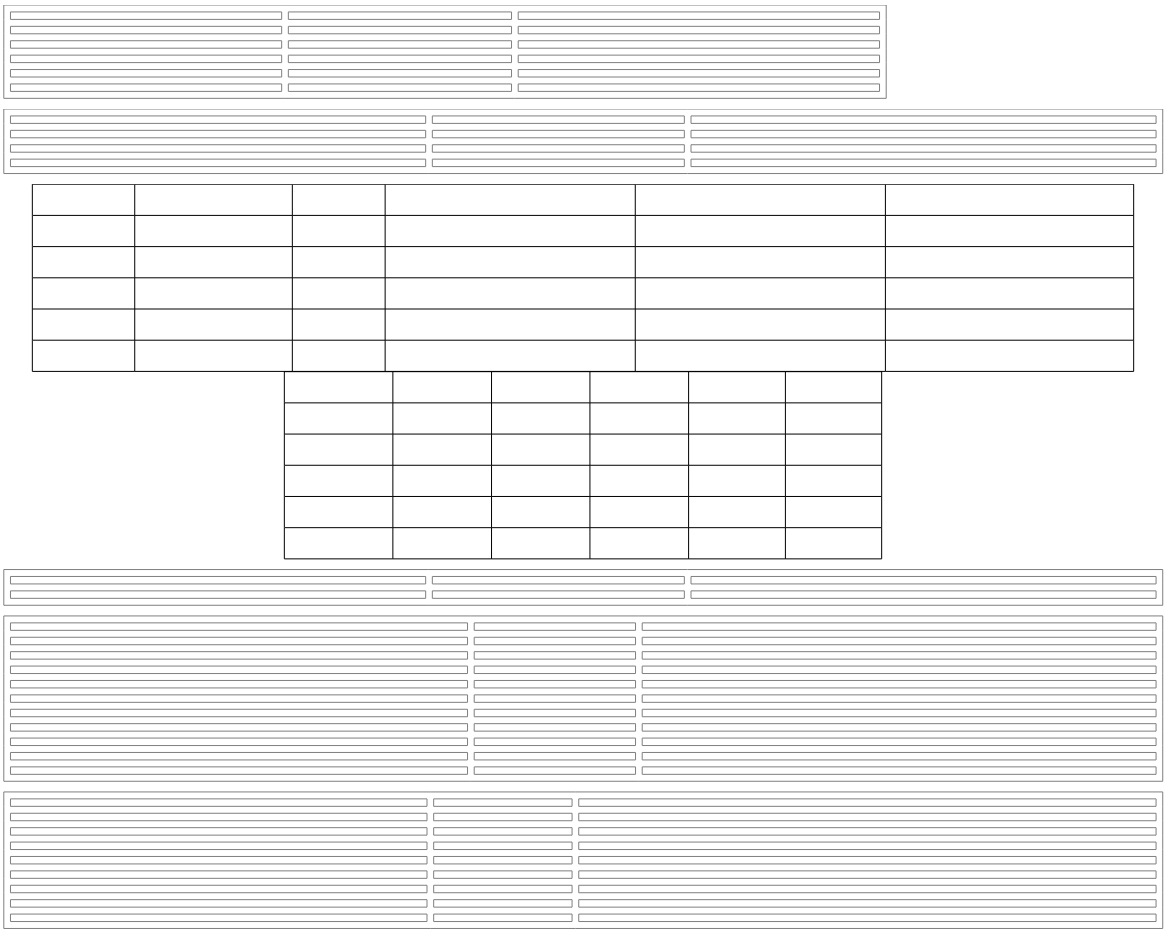
Expression	Result	Comment
=ISBLANK(1)	False	Numbers return false.
=ISBLANK("")	False	Text, even empty string, returns false.
=ISBLANK([.B8])	True	Blank cell is true.
=ISBLANK([.B7])	False	Non-blank cell is false.

See also ISNUMBER, ISTEXT



Copyright © OASIS Open 2006. All Rights Reserved.

Copyright © OASIS Open 2006. All Rights Reserved.



6.14 Logical Functions

The logical functions are the constants TRUE() and FALSE(), the functions that compute logical values NOT(), AND(), and OR(), and the conditional function IF(). The OpenDocument specification mentions "logical operators"; these are simply another name for the logical functions.

Note that because of Error values, any logical function that accepts parameters can actually produce TRUE, FALSE, or an Error value, instead of simply TRUE or FALSE.

These are not bitwise operations, e.g., AND(12;10) produces TRUE(), not 8. See the bit operation functions for bitwise operations.

TBD: Should minimum/maximum number of parameters be specified?

TBD: Should we specify that "If an error value is computed for an expression, then the first error is the result of the logical operation."? Is this widely done ("first error is the result")? Or is there sometimes a pecking order for errors, or is sometimes "last error is the result" the rule?

6.14.1 AND

Summary: Compute logical AND of all parameters.

Syntax: AND({ *Logical*|*NumberSequenceList* L }$^{+}$)

Returns: Logical

Copyright © OASIS Open 2006. All Rights Reserved.

Constraints: Must have 1 or more parameters

Semantics: Computes the logical AND of the parameters. If all parameters are True, returns True; if any are False, returns False. When given one parameter, this has the effect of converting that one parameter into a logical value. When given zero parameters, applications may return a Logical value or an error.

Also in array context a logical AND of all arguments is computed, range or array parameters are not evaluated as a matrix and no array is returned. This behavior is consistent with functions like SUM. To compute a logical AND of arrays per element use the * operator in array context.

Note: Excel 2000, OpenOffice.org 2.0 and Gnumeric 1.4.3 consider AND() with no parameters an error, Kspread 1.4.2 does not.

Test Cases:

Expression	*Result*	*Comment*
=AND(FALSE();FALSE())	False	Simple AND.
=AND(FALSE();TRUE())	False	Simple AND.
=AND(TRUE();FALSE())	False	Simple AND.
=AND(TRUE();TRUE())	True	Simple AND.
=AND(TRUE();NA())	NA	Returns an error if given one.
=AND(1;TRUE())	True	Nonzero considered TRUE.
=AND(0;TRUE())	False	Zero considered FALSE.
=AND(TRUE();TRUE();TRUE())	True	More than two parameters okay.
=AND(TRUE())	True	One parameter okay - simply returns it.

TBD: Implementations of AND and OR do NOT short-circuit on True and False, because they look for error conditions and propagate them. For example, on Excel: AND(FALSE(), #NA) is #NA. Should the "do not short-circuit" rule be REQUIRED? Should short-circuiting be PERMITTED? If short-circuiting is not required, should short-circuiting versions of AND() and OR() be defined, e.g., ANDSC() and ORSC()?

See also OR, IF

6.14.2 FALSE

Summary: Returns constant FALSE

Syntax: FALSE()

Returns: Logical

Constraints: Must have 0 parameters

Semantics: Returns logical constant FALSE. This may be a Number or a distinct type. Syntactically this is a function call, but semantically this is a constant, and applications typically optimize this as a constant.

Test Cases:

Copyright © OASIS Open 2006. All Rights Reserved.

Expression	Result	Comment
=FALSE()	False	Constant
=IF(ISNUMBER(FALSE());FALSE()=0;FALSE())	FALSE	Applications that implement logical values as 0/1 must map TRUE() to 1
2+FALSE()	2	TRUE converts to 1 in Number context

See also TRUE, IF

6.14.3 IF

Summary: Return one of two values, depending on a condition

Syntax: IF(*Logical* Condition [; [*Any* IfTrue] [; [*Any* IfFalse]]])

Returns: Any

Constraints: None.

Semantics: Computes *Condition*. If it is TRUE, it returns *IfTrue*, else it returns *IfFalse*. If there is only 1 parameter, *IfTrue* is considered to be TRUE(). If there are less than 3 parameters, *IfFalse* is considered to be FALSE(). Thus the 1 parameter version converts *Condition* into a Logical value. If there are 2 or 3 parameters but the second parameter is null (two consecutive ;; semicolons), *IfFalse* is considered to be 0. If there are 3 parameters but the third parameter is null, *IfFalse* is considered to be 0. This function only evaluates *IfTrue*, or *ifFalse*, and never both; that is to say, it short-circuits.

Test Cases:

Expression	Result	Comment
=IF(FALSE();7;8)	8	Simple if.
=IF(TRUE();7;8)	7	Simple if.
=IF(TRUE();"HI";8)	="HI"	Can return strings, and the two sides need not have equal types
=IF(1;7;8)	7	A non-zero is considered true.
=IF(5;7;8)	7	A non-zero is considered true.
=IF(0;7;8)	8	A zero is considered false.
=IF(TRUE();[.B4];8)	2	The result can be a reference.
=IF(TRUE();[.B4]+5;8)	7	The result can be a formula.
=IF("x";7;8)	Error	Condition has to be convertible to Logical.
=IF("1";7;8)	Error	Condition has to be convertible to Logical.
=IF("";7;8)	Error	Condition has to be convertible to Logical; empty string is not the same as False

Copyright © OASIS Open 2006. All Rights Reserved.

=IF(FALSE();7)	FALSE	Default IfFalse is FALSE
=IF(3)	TRUE	Default IfTrue is TRUE
=IF(FALSE();7;)	0	Empty parameter is considered 0
=IF(TRUE();;7)	0	Empty parameter is considered 0
=IF(TRUE();4;1/0)	4	If condition is true, ifFalse is not considered – even if it would produce Error.
=IF(FALSE();1/0;5)	5	If condition is false, ifTrue is not considered – even if it would produce Error.

See also AND, OR

6.14.6 NOT

Summary: Compute logical NOT

Syntax: NOT(*Logical* L)

Returns: Logical

Constraints: Must have 1 parameter

Semantics: Computes the logical NOT. If given TRUE, returns FALSE; if given FALSE, returns TRUE.

Test Cases:

Expression	*Result*	*Comment*
=NOT(FALSE())	True	Simple NOT, given FALSE.
=NOT(TRUE())	False	Simple NOT, given TRUE.
=NOT(1/0)	Error	NOT returns an error if given an error value

See also AND, IF

Note: NOT returns an error, if given an error, so unlike some mathematical definitions this can return more than two values: True, False, and Error values.

6.14.7 OR

Summary: Compute logical OR of all parameters.

Syntax: OR({ *Logical*|*NumberSequenceList* L }$^{+}$)

Returns: Logical

Constraints: Must have 1 or more parameters

Semantics: Computes the logical OR of the parameters. If all parameters are False, it **shall** return False; if any are True, it **shall** returns True. When given one parameter, this has the effect

Copyright © OASIS Open 2006. All Rights Reserved.

of converting that one parameter into a logical value. When given zero parameters, applications **may** return a Logical value or an error.

Also in array context a logical OR of all arguments is computed, range or array parameters are not evaluated as a matrix and no array is returned. This behavior is consistent with functions like SUM. To compute a logical OR of arrays per element use the + operator in array context.

Note: Excel 2000, OpenOffice.org 2.0 and Gnumeric 1.4.3 consider OR() with no parameters an error, Kspread 1.4.2 does not.

Test Cases:

Expression	*Result*	*Comment*
=OR(FALSE();FALSE())	False	Simple OR.
=OR(FALSE();TRUE())	True	Simple OR.
=OR(TRUE();FALSE())	True	Simple OR.
=OR(TRUE();TRUE())	True	Simple OR.
=OR(FALSE();NA())	NA	Returns an error if given one.
=OR(FALSE();FALSE();TRUE())	True	More than two parameters okay.
=OR(TRUE())	True	One parameter okay - simply returns it

TBD: Implementations of AND and OR do NOT short-circuit on True and False, because they look for error conditions and propagate them. For example, on Excel: AND(FALSE(), #NA) is #NA. Should the "do not short-circuit" rule be REQUIRED? If it is required, should short-circuiting versions of AND() and OR() be defined, e.g., ANDSC() and ORSC()?

See also AND, IF

6.14.8 TRUE

Summary: Returns constant TRUE

Syntax: TRUE()

Returns: Logical

Constraints: Must have 0 parameters

Semantics: Returns logical constant TRUE. Although this is syntactically a function call, semantically it is a constant, and typical applications optimize this because it is a constant. Note that this may or may not be equal to 1 when compared using “=”. It always has the value of 1 if used in a context requiring Number (because of the automatic conversions), so if ISNUMBER(TRUE()), then it must have the value 1.

Test Cases:

Expression	*Result*	*Comment*
=TRUE()	True	Constant.
=IF(ISNUMBER(TRUE());TRUE()=1;TRUE())	True	Applications that implement logical values as 0/1 must map TRUE() to 1

Copyright © OASIS Open 2006. All Rights Reserved.

2+TRUE()	3	TRUE converts to 1 in Number context

See also FALSE, IF

6.14.9 XOR

Summary: Compute a logical XOR of all parameters.

Syntax: XOR({ *Logical* L }$^{+}$)

Returns: Logical

Constraints: Must have 1 or more parameters.

Semantics: Computes the logical XOR of the parameters such that the result is an addition modulo 2. If an even number of parameters is True it returns False, if an odd number of parameters is True it returns True. When given one parameter, this has the effect of converting that one parameter into a logical value.

Note: The multi-argument form is different from an "exclusive disjunction" operation, which would return True if and only if exactly one argument is True.

Test Cases:

Expression	*Result*	*Comment*
=XOR(FALSE();FALSE())	False	Simple XOR.
=XOR(FALSE();TRUE())	True	Simple XOR.
=XOR(TRUE();FALSE())	True	Simple XOR.
=XOR(TRUE();TRUE())	False	Simple XOR – note that this one is different from OR
=XOR(FALSE();NA())	NA	Returns an error if given one.
=XOR(FALSE();FALSE();TRUE())	True	More than two parameters okay.
=XOR(FALSE(); TRUE();TRUE())	False	More than two parameters okay, and notice that this result is different from OR
=XOR(TRUE(); TRUE();TRUE())	True	More than two parameters okay, the result is ((1 XOR 1) XOR 1), thus a parity.
=XOR(TRUE())	True	One parameter okay - simply returns it

See also AND, OR

Note: Excel 2003, and OpenOffice.org 2.1 don't have XOR(). Gnumeric 1.4.3 considers AND() with no parameters an error, Kspread 1.4.2 does not.

Copyright © OASIS Open 2006. All Rights Reserved.

6.15 Mathematical Functions

#TODO JH: choose a smaller subset of mathematical functions, CONVERT useful? Redefine the NumberSequenceList for SUM etc.#

This section describes functions for various mathematical functions, including trigonometric functions like SIN). Note that the constraint text presumes that a value of type Number is a real number (no imaginary component). Unless noted otherwise, all angle measurements are in radians.

Note: It's debatable if SUM and AVERAGE etc. should be listed under mathematical functions or as statistical functions. COUNT could belong here, under statistics, or under information functions. It doesn't *really* matter; these divisions are arbitrary, simply to make it easier for users to find them.

6.15.1 ABS

Summary: Return the absolute (nonnegative) value.

Syntax: ABS(*Number* N)

Returns: Number

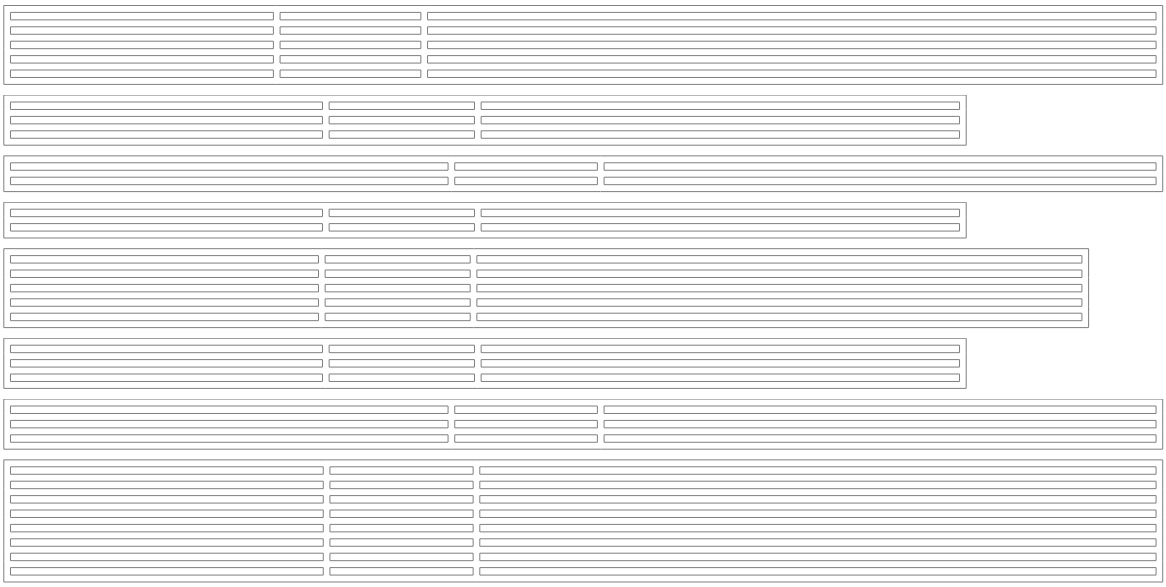
Constraints: None

Semantics: If N < 0, returns -N, otherwise returns N.

Test Cases:

Expression	***Result***	***Comment***
=ABS(-4)	4	If less than zero, return negation
=ABS(4)	4	Positive values return unchanged.

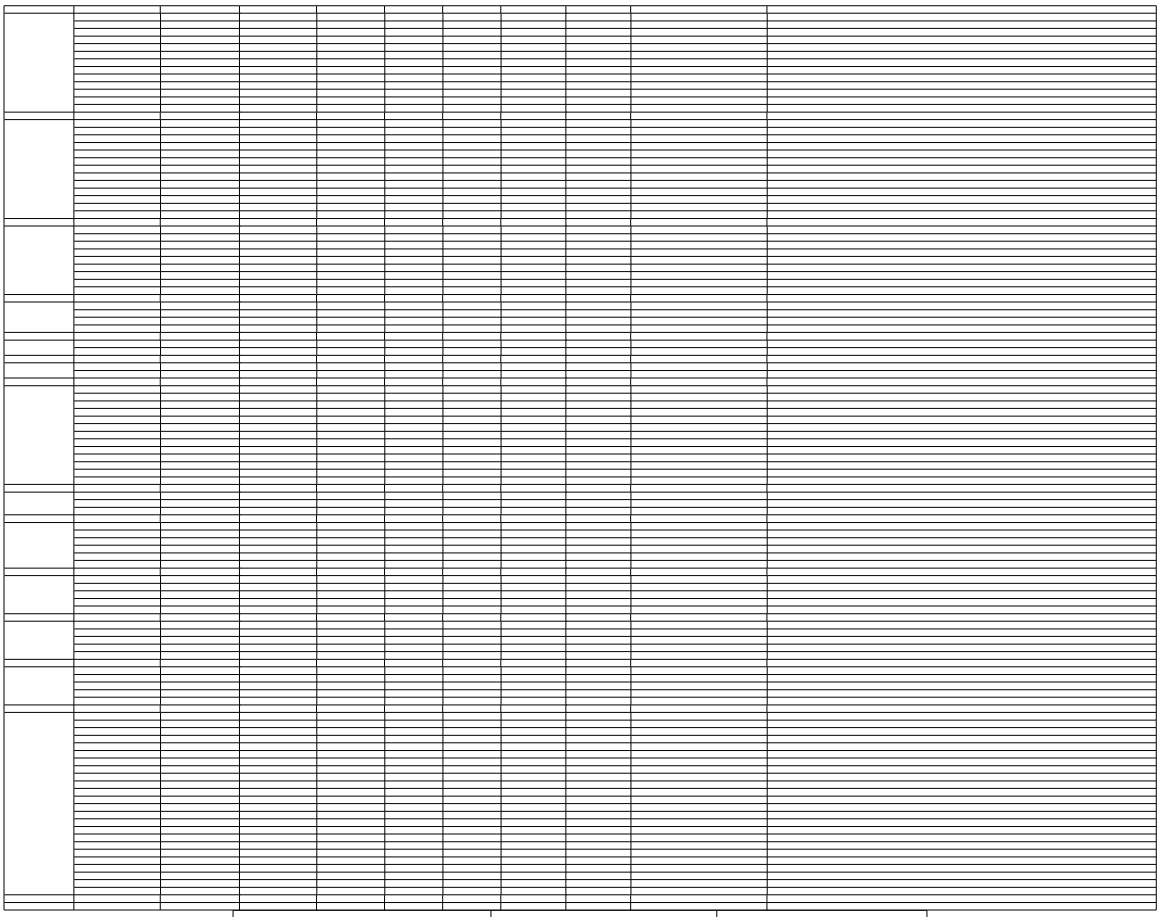
See also Prefix Operator “-”

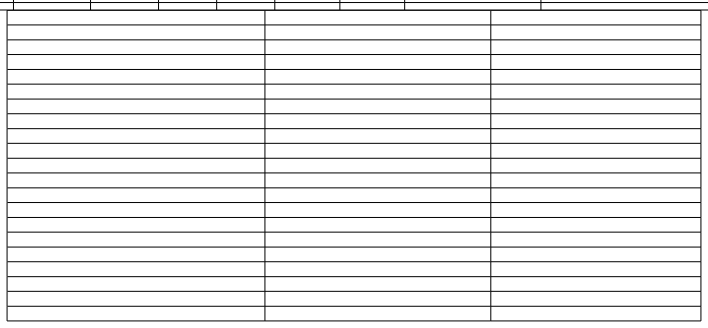


Copyright © OASIS Open 2006. All Rights Reserved.

Copyright © OASIS Open 2006. All Rights Reserved.

Copyright © OASIS Open 2006. All Rights Reserved.





Copyright © OASIS Open 2006. All Rights Reserved.

6.15.18 COS

Summary: Return the cosine of an angle specified in radians.

Syntax: COS(*Number* N)

Returns: Number

Constraints: None

Semantics: Computes the cosine of an angle specified in radians.

$$\cos N = 1 - \frac{N^2}{2!} + \frac{N^4}{4!} - \frac{N^6}{6!} + \ldots$$

Test Cases:

Expression	*Result*	*Comment*
COS(0)	1	COS(0) is 1 – require this result to be exact, since documents may depend on it.
=COS(PI()/4)*2/SQRT(2)	1±ε	cosine of PI()/4 radians is SQRT(2)/2.
=COS(PI()/2)	0±ε	cosine of PI()/2 radians is 0; the test here gives a little "wiggle room" for computational inaccuracies.

See also ACOS, RADIANS, DEGREES

6.15.19 COSH

Summary: Return the hyperbolic cosine of the given hyperbolic angle

Syntax: COSH(*Number* N)

Returns: Number

Constraints: None

Copyright © OASIS Open 2006. All Rights Reserved.

Semantics: Computes the hyperbolic cosine of a hyperbolic angle. The hyperbolic cosine is an analog of the ordinary (circular) cosine. The points (cosh t, sinh t) define the right half of the equilateral hyperbola, just as the points (cos t, sin t) define the points of a circle.

$$\cosh(N) = \frac{e^{N} + e^{-N}}{2}$$

Test Cases:

Expression	*Result*	*Comment*
=COSH(0)	1	Cosh(0) is 0
=COSH(1)	1.5430806348±ε	
=COSH(EXP(1))	7.6101251386±ε	

See also ACOSH, SINH, TANH

6.15.29 EVEN

Summary: Rounds a number up to the nearest even integer. Rounding is away from zero.

Copyright © OASIS Open 2006. All Rights Reserved.

Syntax: EVEN(*Number* N)

Returns: Number

Constraints: None

Semantics: Returns the even integer whose sign is the same as N's and whose absolute value is greater than or equal to the absolute value of N. That is, if rounding is required, it is rounded *away* from zero.

Rationale: There's no need to discuss what happens if N is not a number in this definition; that is handled in the “Conversion to Number” section.

Test Cases:

Expression	*Result*	*Comment*
=EVEN(6)	6	Positive even integers remain unchanged.
=EVEN(-4)	-4	Negative even integers remain unchanged.
=EVEN(1)	2	Non-even positive integers round up.
=EVEN(0.3)	2	Positive floating values round up.
=EVEN(-1)	-2	Non-even negative integers round down.
=EVEN(-0.3)	-2	Negative floating values round down.
=EVEN(0)	0	Since zero is even, EVEN(0) returns zero.

See also ODD

6.15.30 EXP

Summary: Returns e raised by the given number.

Syntax: EXP(*Number* X)

Returns: Number

Constraints: None

Semantics: Computes $e^X = 1 + \frac{X}{1!} + \frac{X^2}{2!} + \frac{X^3}{3!} + \frac{X^n}{n!} + \ldots$

Note: Test case 3 implies a very minimal precision requirement for e (i.e. Exp(1)), but this is a trivial requirement and clearly higher accuracy is better.

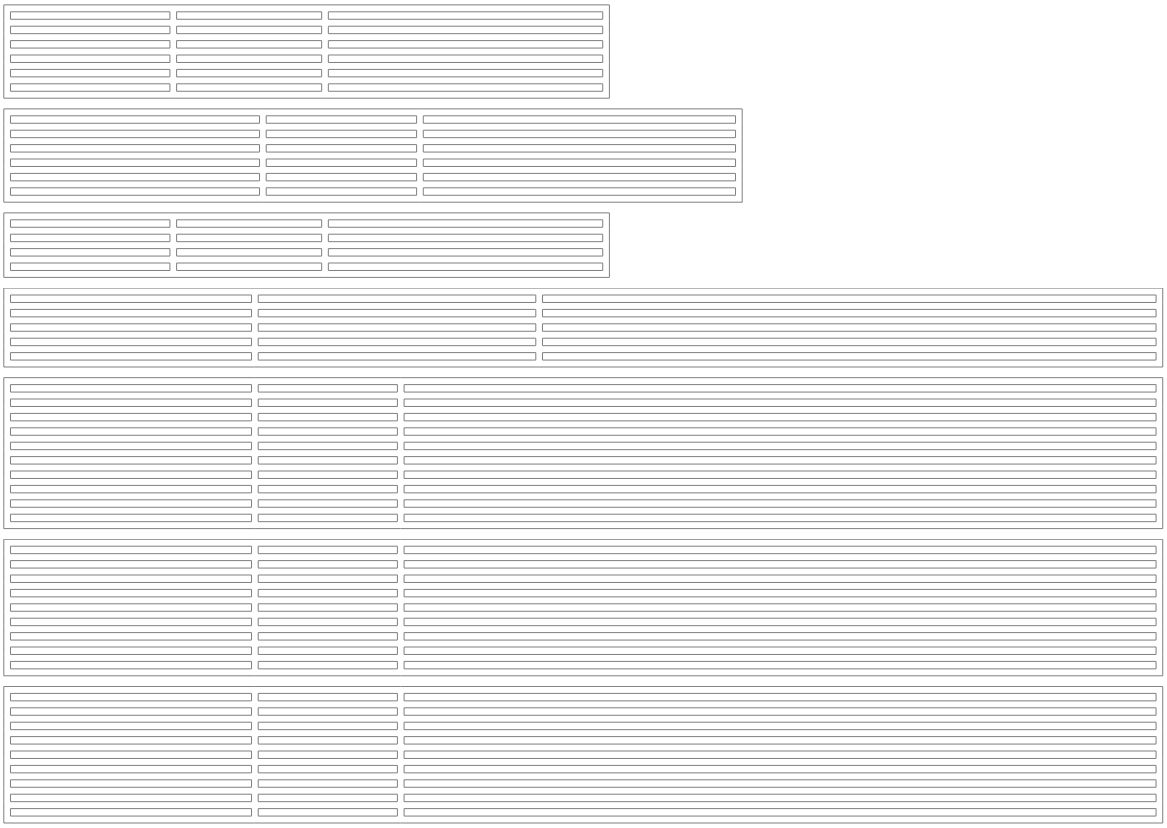
Test Cases:

Expression	*Result*	*Comment*
=EXP(0)	1	Anything raised to the 0 power is 1.
=EXP(LN(2))	2±ε	The EXP function is the inverse of the LN function.

Copyright © OASIS Open 2006. All Rights Reserved.

=EXP(1)	2.71828182845904523536 ±ε	The value of the natural logarithm e.

See also LOG, LN



6.15.38 LN

Summary: Return the natural logarithm of a number.

Syntax: LN(*Number* X)

Returns: Number

Constraints: X>0

Semantics: Computes the natural logarithm (base e) of the given number.

$$\ln x = 2\left[\frac{x-1}{x+1} + \frac{1}{3}\left(\frac{x-1}{x+1}\right)^3 + \frac{1}{5}\left(\frac{x-1}{x+1}\right)^5 + \ldots\right]$$

Test Cases:

Expression	*Result*	*Comment*
=LN(1)	0	The logarithm of 1 (in any base) is 0.

Copyright © OASIS Open 2006. All Rights Reserved.

=LN(EXP(1))	1	The natural logarithm of e is 1.
=LN(20)	2.995732274±ε	Trivial test
=LN(0.2)	-1.609437912±ε	This tests a value between 0 and 0.5. Values in this domain are valid, but implementations that compute LN(x) by blindly summing the series (1/n)((x-1)/x)^n won't get this value correct, because that series requires x > 0.5.
=LN(0)	Error	The argument must be greater than zero.
=LN([.B7])	Error	The natural logarithm of a non-number gives an error.

See also LOG, LOG10, POWER, EXP

6.15.39 LOG

Summary: Return the logarithm of a number in a specified base.

Syntax: LOG(*Number* N [; *Number* Base = 10])

Returns: Number

Constraints: N > 0

Semantics: Computes the logarithm of a number in the specified base. Note that if the base is not specified, the logarithm base 10 is returned.

Test Cases:

Expression	*Result*	*Comment*
=LOG(1; 10)	0	The logarithm of 1 (in any base) is 0.
=LOG(1;EXP(1))	0	The natural logarithm of 1 is 0.
=LOG(10;10)	1	The base 10 logarithm of 10 is 1.
=LOG(10)	1	If the base is not specified, base 10 is assumed.
=LOG(8*8*8;8)	3	Log base 8 of 8^3 should return 3.
=LOG(0;10)	Error	The argument must be greater than zero.

See also LOG10, LN, POWER, EXP

Note: OOo2 requires the base for log, instead of allowing it to be optional. Excel, etc. consider them optional. Making LOG's second argument optional will require a change to OOo.

6.15.40 LOG10

Summary: Return the base 10 logarithm of a number.

Copyright © OASIS Open 2006. All Rights Reserved.

Syntax: LOG10(*Number* N)

Returns: Number

Constraints: N > 0

Semantics: Computes the base 10 logarithm of a number.

Test Cases:

Expression	*Result*	*Comment*
=LOG10(1)	0	The logarithm of 1 (in any base) is 0.
=LOG10(10)	1	The base 10 logarithm of 10 is 1.
=LOG10(100)	2	The base 10 logarithm of 100 is 2.
=LOG10(0)	Error	The argument must be greater than zero.
=LOG10("H")	Error	The logarithm of a non-number gives an error.

See also LOG, LN, POWER, EXP

6.15.41 MOD

Summary: Return the remainder when one number is divided by another number.

Syntax: MOD(*Number* a ; *Number* b)

Returns: Number

Constraints: b != 0

Semantics: Computes the remainder of a/b. The remainder has the same sign as *b*.

Test Cases:

Expression	*Result*	*Comment*
=MOD(10;3)	1	10/3 has remainder 1.
=MOD(2;8)	2	2/8 is 0 remainder 2.
=MOD(5.5;2.5)	0.5	The numbers need not be integers.
=MOD(-2;3)	1	The location of the sign matters.
=MOD(2;-3)	-1	The location of the sign matters.
=MOD(-2;-3)	-2	The location of the sign matters.
=MOD(10;0)	Error	Division by zero is not allowed

Note: MOD with negative numbers is tricky. The tests for MOD(-2;3), MOD(2;-3), and MOD(-2;-3) work with Gnumeric, OOo2, and Excel 2002 as shown. KSpread 1.4.2 also computes MOD(-2;3) as 1. However, KSpread 1.4.2 (on Fedora Core 4) computed MOD(-2;-3) as -5 (not -1), and MOD(2;-3) computes as 2 (not -1). On January 9, 2006, KSpread developer Tomas Mecir

Copyright © OASIS Open 2006. All Rights Reserved.

announced that KSpread's development version had been changed, and the next release of KSpread would produce the same results with these other negative values.

See also Infix Operator "/", QUOTIENT

6.15.43 ODD

Summary: Rounds a number up to the nearest odd integer, where "up" means "away from 0".

Syntax: ODD(*Number* N)

Returns: Number

Constraints: None

Semantics: Returns the odd integer whose sign is the same as N's and whose absolute value is greater than or equal to the absolute value of N. In other words, any "rounding" is away from zero. By definition, ODD(0) is 1.

Rationale: There's no need to discuss what happens if N is not a number in this definition; that is handled in the “Conversion to Number” section.

Test Cases:

Expression	***Result***	***Comment***
=ODD(5)	5	Positive odd integers remain unchanged.
=ODD(-5)	-5	Negative odd integers remain unchanged.
=ODD(2)	3	Non-odd positive integers round up.
=ODD(0.3)	1	Positive floating values round up.
=ODD(-2)	-3	Non-odd negative integers round down.
=ODD(-0.3)	-1	Negative floating values round down.
=ODD(0)	1	By definition, ODD(0) is 1.

See also EVEN

6.15.44 PI

Summary: Return the approximate value of Pi.

Syntax: PI()

Returns: Number

Constraints: None.

Semantics: This function takes no arguments and returns the (approximate) value of pi. Pi is an irrational number, and cannot be represented by a finite number of digits. Implementations are not required to represent pi exactly (obviously), but they **should** make an attempt to give an accurate

Copyright © OASIS Open 2006. All Rights Reserved.

result given their numerical representation method. Implementations which use a fixed-length numerical representation **should** use the closest possible numerical representation.

Note: The test case below implies a very minimal precision requirement for e (i.e. Exp(1)), but this is a trivial requirement and clearly higher accuracy is better.

Test Cases:

Expression	*Result*	*Comment*
=PI()	3.1415926 535897932 384626433 8327950±ε	The approximate value of pi. Lots of digits given here, in case the implementation can actually handle that many, but implementations are *not* required to exactly store this many digits. PI() **should** return the closest possible numeric representation in a fixed-length representation.

See also SIN, COS

6.15.45 POWER

Summary: Return the value of one number raised to the power of another number.

Syntax: POWER(*Number* a ; *Number* b)

Returns: Number

Constraints: None

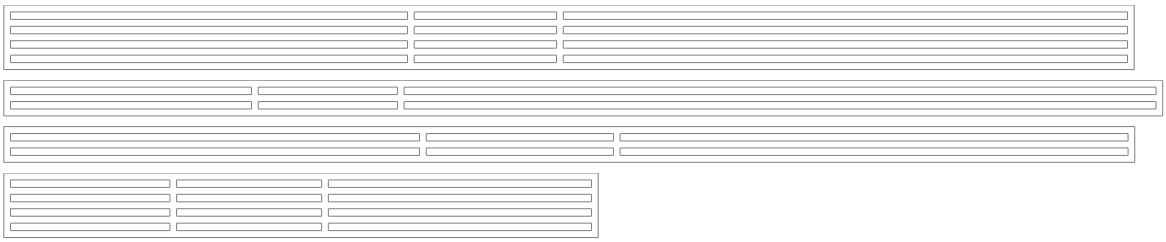
Semantics: Computes a raised to the power b.

Test Cases:

Expression	*Result*	*Comment*
=POWER(10;0)	1	Anything raised to the 0 power is 1.
=POWER(2;8)	256	2^8 is 256.

See also LOG, LOG10, LN, EXP

Copyright © OASIS Open 2006. All Rights Reserved.



6.15.54 SIN

Summary: Return the sine of an angle specified in radians

Syntax: SIN(*Number* N)

Returns: Number

Constraints: None

Semantics: Computes the sine of an angle specified in radians.

$$\sin(N) = N - \frac{N^3}{3!} + \frac{N^5}{5!} - \frac{N^7}{7!} + \ldots$$

Test Cases:

Expression	*Result*	*Comment*
=SIN(0)	0	Sine of 0 is 0; this is required to be exact, since some documents may depend on getting this identity exactly correct.
=SIN(PI()/4.0)*2/SQRT(2)	1±ε	sine of PI()/4 radians is SQRT(2)/2.
=SIN(PI()/2.0)	1±ε	sine of PI()/2 radians is 1.

See also ASIN, RADIANS, DEGREES

6.15.55 SINH

Summary: Return the hyperbolic sine of the given hyperbolic angle

Syntax: SINH(*Number* N)

Returns: Number

Constraints: None

Semantics: Computes the hyperbolic sine of a hyperbolic angle. The hyperbolic sine is an analog of the ordinary (circular) sine. The points (cosh t, sinh t) define the right half of the equilateral hyperbola, just as the points (cos t, sin t) define the points of a circle.

$$\sinh(N) = \frac{e^N - e^{-N}}{2}$$

Test Cases:

Copyright © OASIS Open 2006. All Rights Reserved.

Expression	Result	Comment
=SINH(0)	0	Sinh(0) is 0
=SINH(1)	1.1752011936±ε	
=SINH(EXP(1))	7.5441371028±ε	

See also ASINH, COSH, TANH

6.15.57 SQRT

Summary: Return the square root of a number

Syntax: SQRT(*Number* N)

Returns: Number

Constraints: N>=0

Semantics: Returns the square root of a non-negative number. This function must produce an error if given a negative number; for producing complex numbers, see IMSQRT.

Test Cases:

Expression	Result	Comment
=SQRT(4)	2	The square root of 4 is 2.
=SQRT(-4)	Error	The argument must be non-negative

See also POWER, IMSQRT, SQRTPI

6.15.60 SUM

Summary: Sum (add) the set of numbers, including all numbers in ranges

Syntax: SUM({ *NumberSequenceList* N }$^{+}$)

Returns: Number

Constraints: None

Semantics: Adds numbers (and only numbers) together (see the text on conversions). Applications may allow SUM to receive 0 parameters (and return 0), but portable documents **must not** depend on SUM() with zero parameters returning 0.

Copyright © OASIS Open 2006. All Rights Reserved.

Test Cases:

Expression	*Result*	*Comment*
=SUM(1;2;3)	6	Simple sum.
=SUM(TRUE();2;3)	6	TRUE() is 1.
=SUM([.B4:.B5])	5	2+3 is 5.

See also AVERAGE

Note: The exact semantics of SUM() in Excel are defined in (Walkenbach, 2004, pg 705). Lotus 1-2-3v9.8.1 does not accept SUM with no parameters.

6.15.64 TAN

Summary: Return the tangent of an angle specified in radians

Syntax: TAN(*Number* N)

Returns: Number

Constraints: None

Semantics: Computes the tangent of an angle specified in radians.

```
TAN(x) = SIN(x) / COS(x)
```

Test Cases:

Expression	*Result*	*Comment*
=TAN(PI()/4.0)	1±ε	tangent of PI()/4.0 radians.

See also ATAN, ATAN2, RADIANS, DEGREES, SIN, COS, COT

6.15.65 TANH

Summary: Return the hyperbolic tangent of the given hyperbolic angle

Syntax: TANH(*Number* N)

Returns: Number

Constraints: None

Copyright © OASIS Open 2006. All Rights Reserved.

Semantics: Computes the hyperbolic tangent of a hyperbolic angle. The hyperbolic tangent is an analog of the ordinary (circular) tangent. The points (cosh t, sinh t) define the right half of the equilateral hyperbola, just as the points (cos t, sin t) define the points of a circle.

$$\tanh(N) = \frac{\sinh(N)}{\cosh(N)} = \frac{e^{N} - e^{-N}}{e^{N} + e^{-N}}$$

Test Cases:

Expression	*Result*	*Comment*
=TANH(0)	0	tanh(0) is 0
=TANH(1)	0.76159415 5955765±ε	
=TANH(-1)	-0.76159415 5955765±ε	
=TANH(EXP(1))	0.99132891 58±ε	

See also ATANH, SINH, COSH

6.16 Rounding Functions

#TODO JH: define a smaller subset#

The following are various “rounding” functions that convert an arbitrary Number into an Integer (a subset of possible Number values).

Note: Typical implementations of rounding functions first “round” the internal number representation to the nearest integer if it is numerically adjacent to it in the numerical representation, *then* they perform rounding. Thus, INT((1/3)*3) becomes 1. Some of these functions have bizarre semantics.

6.16.1 CEILING

Summary: Round a number *N* **up** to the nearest multiple of the second parameter, *significance*.

Syntax: CEILING(*Number* N [; *Number* significance [; *Number* mode]])

Returns: Number

Constraints: Both *N* and *significance* must be numeric and have the same sign if not 0.

Semantics: Rounds a number up to a multiple of the second number. If *significance* is omitted it is assumed to be -1 if *N* is negative and +1 if *N* is non-negative, making the function act like the normal mathematical *ceiling* function. If *mode* is given and not equal to zero, the amount of *N* is rounded away from zero to a multiple of *significance* and then the sign applied. If mode is omitted or zero, rounding is toward positive infinity; the number is rounded to the smallest multiple of significance that is equal-to or greater than N. If any of the two parameters *N* or *significance* is zero, the result is zero.

Copyright © OASIS Open 2006. All Rights Reserved.

Warning: Many application user interfaces have a CEILING function with only two parameters, and somewhat different semantics than given here (e.g., they operate as if there was a non-zero mode value). These CEILING functions are inconsistent with the standard mathematical definition of CEILING. Such applications **shall** convert such formulas into a 3-parameter format when saving in OpenFormula format, so that the saved formulas work correctly.

Test Cases:

Expression	*Result*	*Comment*
=CEILING(2; 1)	2	Rounding an integer returns the integer.
=CEILING(2; 3)	3	Round 2 up to the nearest multiple of 3.
=CEILING(2.5; 1)	3	Round 2.5 up to the next larger integer.
=CEILING(2.5; 2)	4	Round 2.5 up to the nearest multiple of 2.
=CEILING(2.5; 2.2)	4,4	Round 2.5 up to the nearest multiple of 2.2.
=CEILING(-2.5;1)	Error	Sign of number and significance must match.
=CEILING(-2.5; -1)	-2	This is a problem case, different programs get different results. Some get -3, rounding away from zero instead toward a ceiling.
=CEILING(-2.5; -1;1)	-3	Round away from zero if third parameter given and not 0.
=CEILING(-2.5;0)	0	Rounding any value to a multiple of 0 results in 0.
=CEILING(0;-1)	0	Rounding 0 to any multiple results in 0.
=CEILING(-1.1)	-1	Single-argument version works like standard mathematical function.

See also FLOOR, INT

Note: In Gnumeric, Kspread and Excel, CEILING(-2.5;-1) gives -3, in OOo it gives -2. Since the function is supposed to round up, -2 seems like the correct answer. This function follows the strange *round away from zero* of Excel only if the third parameter is given.

Note: Kspread 1.5.0 does not pass the CEILING(x;0) test case and generates an error instead.

Note: Excel 2003 has *only* a two-parameter CEILING, with very odd semantics that are different from the standard mathematical meaning. In Excel, CEILING(-2.5; -1) gives -3, not -2. The definition here requires that some applications translate their CEILING (and FLOOR).

6.16.2 INT

Summary: Rounds a number down to the nearest integer.

Syntax: INT(*Number* N)

Returns: Number

Constraints: None

Semantics: Returns the nearest integer whose value is less than or equal to N. Rounding is towards negative infinity.

Copyright © OASIS Open 2006. All Rights Reserved.

Test Cases:

Expression	*Result*	*Comment*
=INT(2)	2	Positive integers remain unchanged
=INT(-3)	-3	Negative integers remain unchanged
=INT(1.2)	1	Positive floating values are truncated
=INT(1.7)	1	It doesn't matter if the fractional part is > 0.5
=INT(-1.2)	-2	Negative floating values round towards negative infinity
=INT((1/3)*3)	1	Naive users expect INT to "correctly" make integers even if there are limits on precision.

See also ROUND, TRUNC

6.16.3 FLOOR

Summary: Round a number *N* **down** to the nearest multiple of the second parameter, *significance*.

Syntax: FLOOR(*Number* N [; *Number* significance [; *Number* mode]])

Returns: Number

Constraints: Both *N* and *significance* must be numeric and have the same sign.

Semantics: Rounds a number down to a multiple of the second number. If *significance* is omitted it is assumed to be -1 if *N* is negative and +1 if *N* is positive, making the function act like the normal mathematical *floor* function. If *mode* is given and not equal to zero, the amount of *N* is rounded toward zero to a multiple of *significance* and then the sign applied. Otherwise, it rounds toward negative infinity, and the result is the largest multiple of significance that is less than or equal to N. If any of the two parameters *N* or *significance* is zero, the result is zero.

Warning: Many application user interfaces have a FLOOR function with only two parameters, and somewhat different semantics than given here (e.g., they operate as if there was a non-zero mode value). These FLOOR functions are inconsistent with the standard mathematical definition of FLOOR. Such applications **shall** convert such formulas into a 3-parameter format when saving in OpenFormula format, so that the saved formulas work correctly.

Test Cases:

Expression	*Result*	*Comment*
=FLOOR(2; 1)	2	Rounding an integer returns the integer.
=FLOOR(2.5; 1)	2	Round 2.5 down to the next smaller integer.
=FLOOR(5; 2)	4	Round 5 down to the nearest multiple of 2.
=FLOOR(5; 2.2)	4,4	Round 5 down to the nearest multiple of 2.2.
=FLOOR(-2.5;1)	Error	Sign of number and significance must match.
=FLOOR(-2.5; -1)	-3	This is a problem case, different programs get different results.

Copyright © OASIS Open 2006. All Rights Reserved.

		Some get -2, rounding toward zero instead toward a floor.
=FLOOR(-2.5; -1;1)	-2	Round toward zero if third parameter given and not 0.
=FLOOR(-2.5;0)	0	Rounding any value to a multiple of 0 results in 0.
=FLOOR(0;-1)	0	Rounding 0 to any multiple results in 0.
=FLOOR(-1.1)	-2	Rounds down toward negative infinity with one argument

See also CEILING, INT

Note: In Gnumeric and Excel, FLOOR(-2.5;-1) gives -2, in OOo it gives -3. Since the function is supposed to round down, -3 seems like the correct answer. Kspread FLOOR(-2.5) gives -3. This function follows the strange *round toward zero* only if the third parameter is given.

Note: Kspread 1.5.0 implements FLOOR with only one parameter.

6.16.4 MROUND

Summary: Rounds the number to given multiple.

Syntax: MROUND(*Number* a ; *Number* b)

Returns: Number

Constraints: None

Semantics: Returns the number X, for which the following holds: X/b=INT(X/b) (b divides X), and for any other Y with the same property, ABS(Y-a)>=ABS(X-a). In case that two such X exist, the greater one is the result. In less formal language, this function rounds the number *a* to multiples of *b*.

Test Cases:

Expression	*Result*	*Comment*
=MROUND(1564;100)	1600	Rounding up.
=MROUND(1520;100)	1500	Rounding down.
=MROUND(1550;100)	1600	Exactly at .5, rounding up.
=MROUND(41.89;8)	40	Can round to any multiples.

See also ROUND

6.16.5 ROUND

Summary: Rounds the value X to the nearest multiple of the power of 10 specified by Digits.

Syntax: ROUND(*Number* X [; *Number* Digits = 0])

Returns: Number

Constraints: None

Copyright © OASIS Open 2006. All Rights Reserved.

Semantics: Round number X to the precision specified by Digits. The number X is rounded to the nearest power of 10 given by $10^{-Digits}$. If Digits is zero, or absent, round to the nearest decimal integer. If Digits is non-negative, round to the specified number of decimal places. If Digits is negative, round to the left of the decimal point by -Digits places. If X is halfway between the two nearest values, the result must round away from zero. Note that if X is a Number, and Digits <= 0, the results will always be an integer (without a fractional component).

Test Cases:

Expression	*Result*	*Comment*
=ROUND(10.1; 0)	10	If b is not specified, round to the nearest integer.
=ROUND(9.8;0)	10	Round to nearest value (different than INT)
=ROUND(0.5; 0)	1	0.5 rounds up, away from zero.
=ROUND(1/3;0)	0	Round to the nearest integer.
=ROUND(1/3;1)	0.3	Round to one decimal place.
=ROUND(1/3;2)	0.33	Round to two decimal places.
=ROUND(1/3;2.9)	0.33	If b is not an integer, it is truncated.
=ROUND(5555;-1)	5560	Round to the nearest 10.
=ROUND(-1.1; 0)	-1	Negative number rounded to the nearest integer
=ROUND(-1.5; 0)	-2	Negative number rounds away from zero
=ROUND(-1.5)	-2	Default precision is 0
=ROUND(1.1)	1	
=ROUND(9.8)	10	

See also TRUNC, INT

Note: Excel 2003 requires both parameters. Many other implementations, such as OpenOffice.org, make the second parameter optional (with a default of 0).

6.16.6 ROUNDDOWN

Summary: Rounds the value X down to the nearest multiple of the power of 10 specified by Digits

Syntax: ROUNDDOWN(*Number* X [; *Number* Digits = 0])

Returns: Number

Constraints: None

Semantics: Round X down, to the precision specified by Digits. The number returned is the largest number that does not exceed X and is a multiple of $10^{-Digits}$. If Digits is zero, or absent, round to the nearest decimal integer. If Digits is positive, round to the specified number of decimal places. If Digits is negative, round to the left of the decimal point by -Digits places.

Test Cases:

Copyright © OASIS Open 2006. All Rights Reserved.

Expression	Result	Comment
=ROUNDDOWN(1.45673; -2)	1.45	
=ROUNDDOWN(1; 0)	1	
=ROUNDDOWN(1)	1	
=ROUNDDOWN(9; 1)	0	
=ROUNDDOWN(9; 0)	9	
=ROUNDDOWN(-1.1)	-2	
=ROUNDDOWN(-1.9)	-2	

See also TRUNC, INT, ROUND

Note: Excel 2003 requires both parameters. Many other implementations, such as OpenOffice.org, make the second parameter optional (with a default of 0).

6.16.7 ROUNDUP

Summary: Rounds the value X up to the nearest multiple of the power of 10 specified by Digits

Syntax: ROUNDUP(*Number* X [; *Number* Digits = 0])

Returns: Number

Constraints: None

Semantics: Round X up, to the precision specified by Digits. The number returned is the smallest number that is not less than X and is a multiple of 10^{Digits}. If Digits is zero, or absent, round to the smallest decimal integer larger or equal to X. If Digits is positive, round to the specified number of decimal places. If Digits is negative, round to the left of the decimal point by -Digits places.

Test Cases:

Expression	Result	Comment
=ROUNDUP(1.45673; -2)	1.46	
=ROUNDUP(1.1;0)	2	
=ROUNDUP(1.9;0)	2	
=ROUNDUP(1)	1	
=ROUNDUP(9;1)	10	

Copyright © OASIS Open 2006. All Rights Reserved.

=ROUNDUP(9;0)	9	
=ROUNDUP(-1.1)	-1	
=ROUNDUP(-1.9)	-1	

See also TRUNC, INT, ROUND

Note: Excel 2003 requires both parameters. Many other implementations, such as OpenOffice.org, make the second parameter optional (with a default of 0).

6.16.8 TRUNC

Summary: Truncate a number to a specified number of digits.

Syntax: TRUNC(*Number* a ; *Number* b)

Returns: Number

Constraints: None

Semantics: Truncate number a to the number of digits specified by b. If b is zero, or absent, truncate to a decimal integer. If b is positive, truncate to the specified number of decimal places. If b is negative, truncate to the left of the decimal point. If b is not an integer, it is truncated.

Test Cases:

Expression	*Result*	*Comment*
=TRUNC(10.1)	10	If b is not specified, truncate to the nearest integer.
=TRUNC(0.5)	0	Truncate rather than rounding.
=TRUNC(1/3;0)	0	Truncate to an integer.
=TRUNC(1/3;1)	0.3	Truncate to one decimal place.
=TRUNC(1/3;2)	0.33	Truncate to two decimal places.
=TRUNC(1/3;2.9)	0.33	If b is not an integer, it is truncated.
=TRUNC(5555;-1)	5550	Truncate to the nearest 10.
=TRUNC(-1.1)	-1	Negative number truncated to an integer
=TRUNC(-1.5)	-1	Negative number truncated to an integer

See also ROUND, INT

6.17 Statistical Functions

#TODO JH: define a smaller subset, distribution function useful?#

The following are statistical functions (functions that report information on a set of numbers). Some functions that could also be considered statistical functions, such as SUM, are listed elsewhere.

Copyright © OASIS Open 2006. All Rights Reserved.

TODO: The LEGACY.* functions haven't grown an equivalent function name.. drop the prefix?

6.17.2 AVERAGE

Summary: Average the set of numbers

Syntax: AVERAGE({ *NumberSequence* N }$^{+}$)

Returns: Number

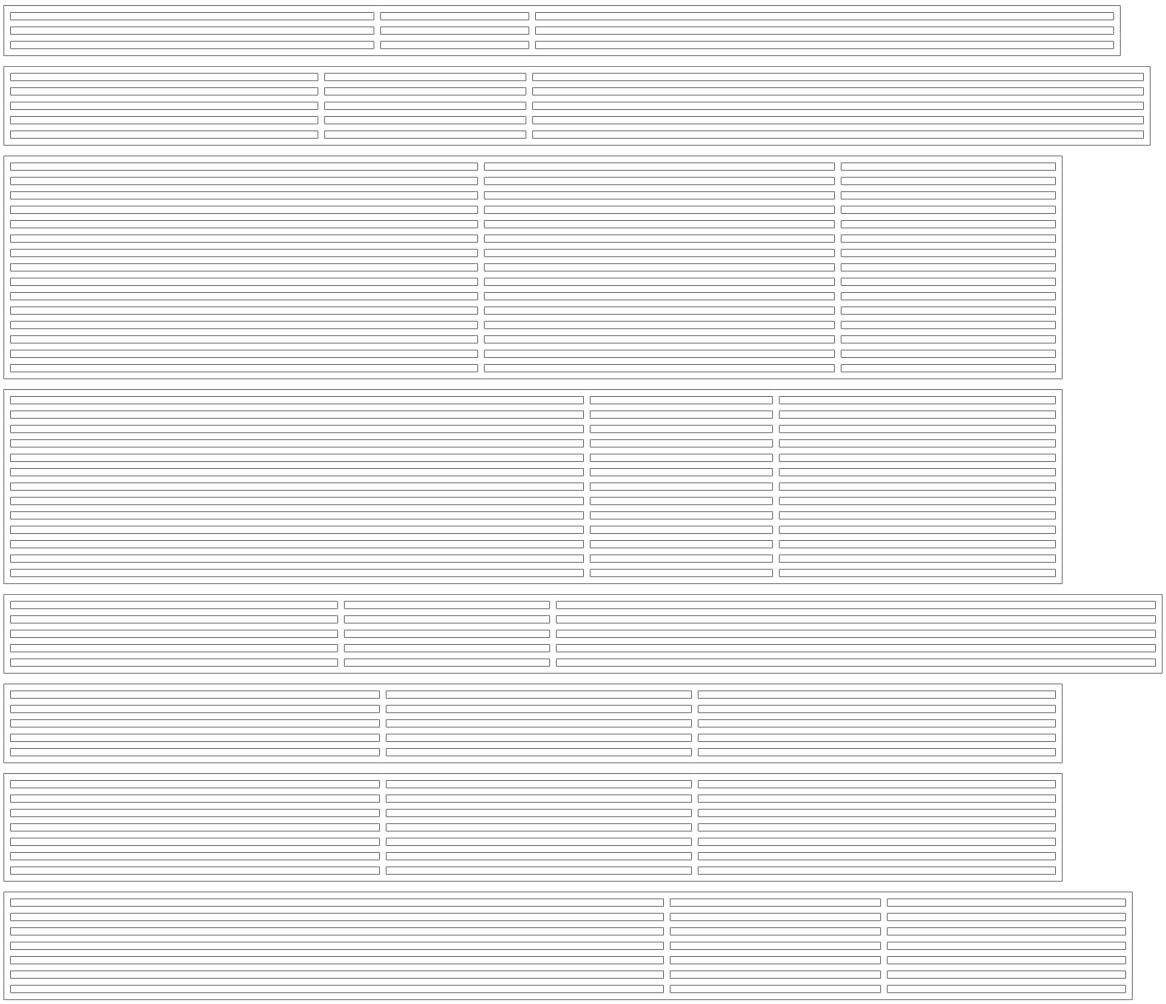
Constraints: At least one number included. Returns an error if no numbers provided.

Semantics: Computes SUM(*List*) / COUNT(*List*).

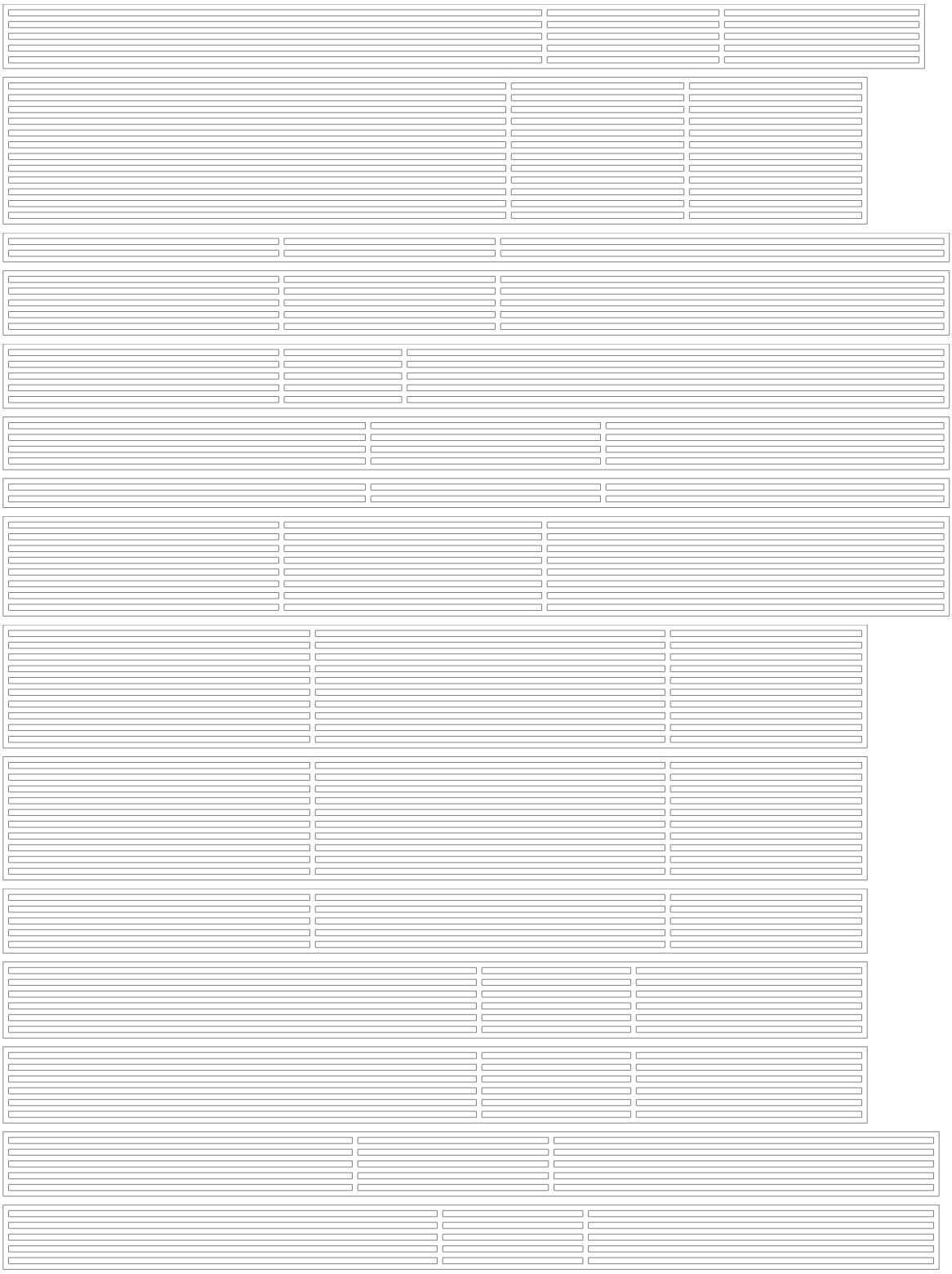
Test Cases:

Expression	*Result*	*Comment*
=AVERAGE(2;4)	3	Simple average.

See also SUM, COUNT



Copyright © OASIS Open 2006. All Rights Reserved.



6.17.27 FORECAST

Summary: Extrapolates future values based on existing x and y values.

Copyright © OASIS Open 2006. All Rights Reserved.

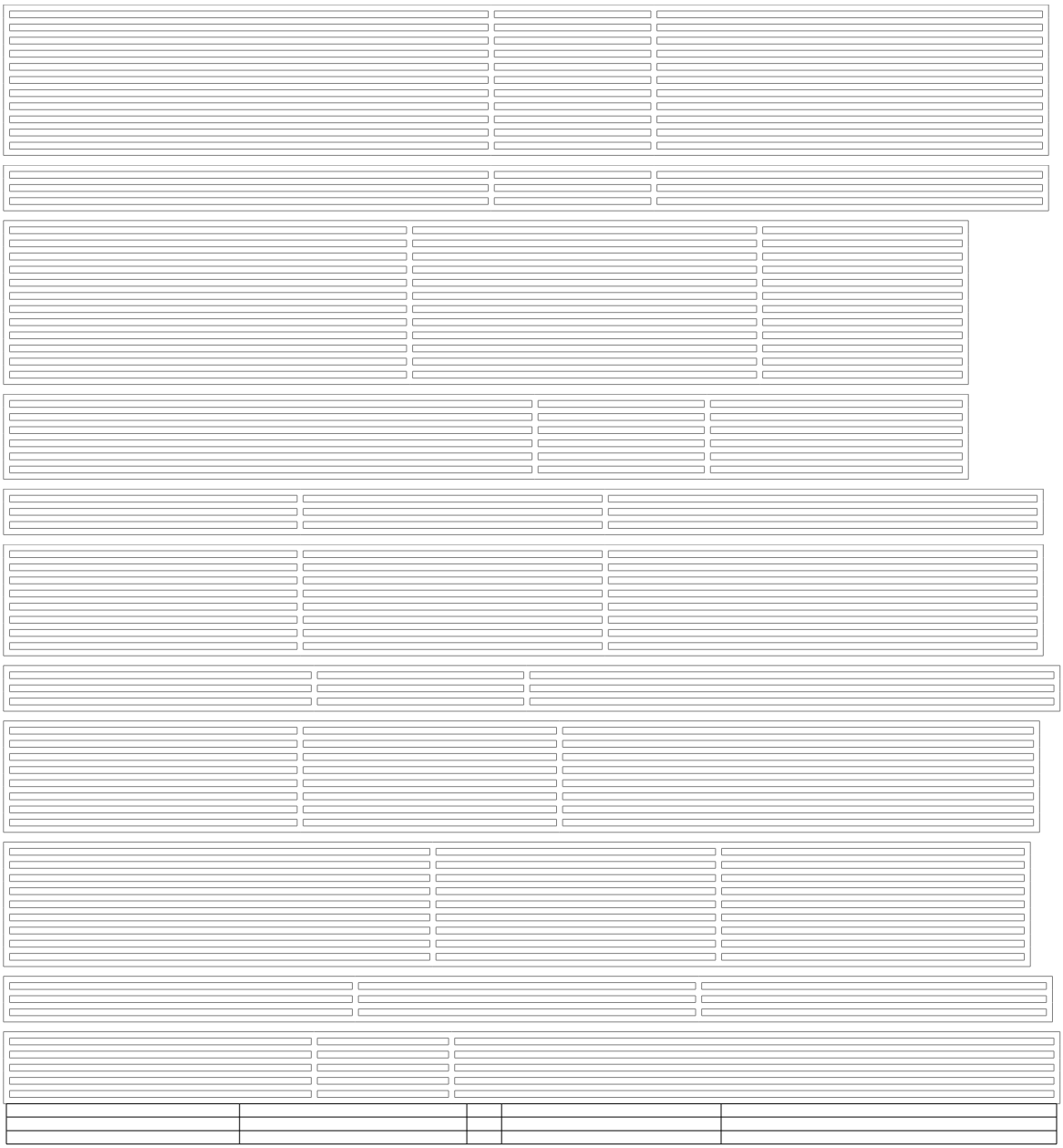
Syntax: FORECAST(*Number* Value ; *ForceArray NumberSequence* Data_Y ; *ForceArray NumberSequence* Data_X)

Returns: Number

Semantics:
Value is the x-value, for which the y-value on the linear regression is to be returned.
Data_Y is the array or range of known y-values.
Data_X is the array or range of known x-values.

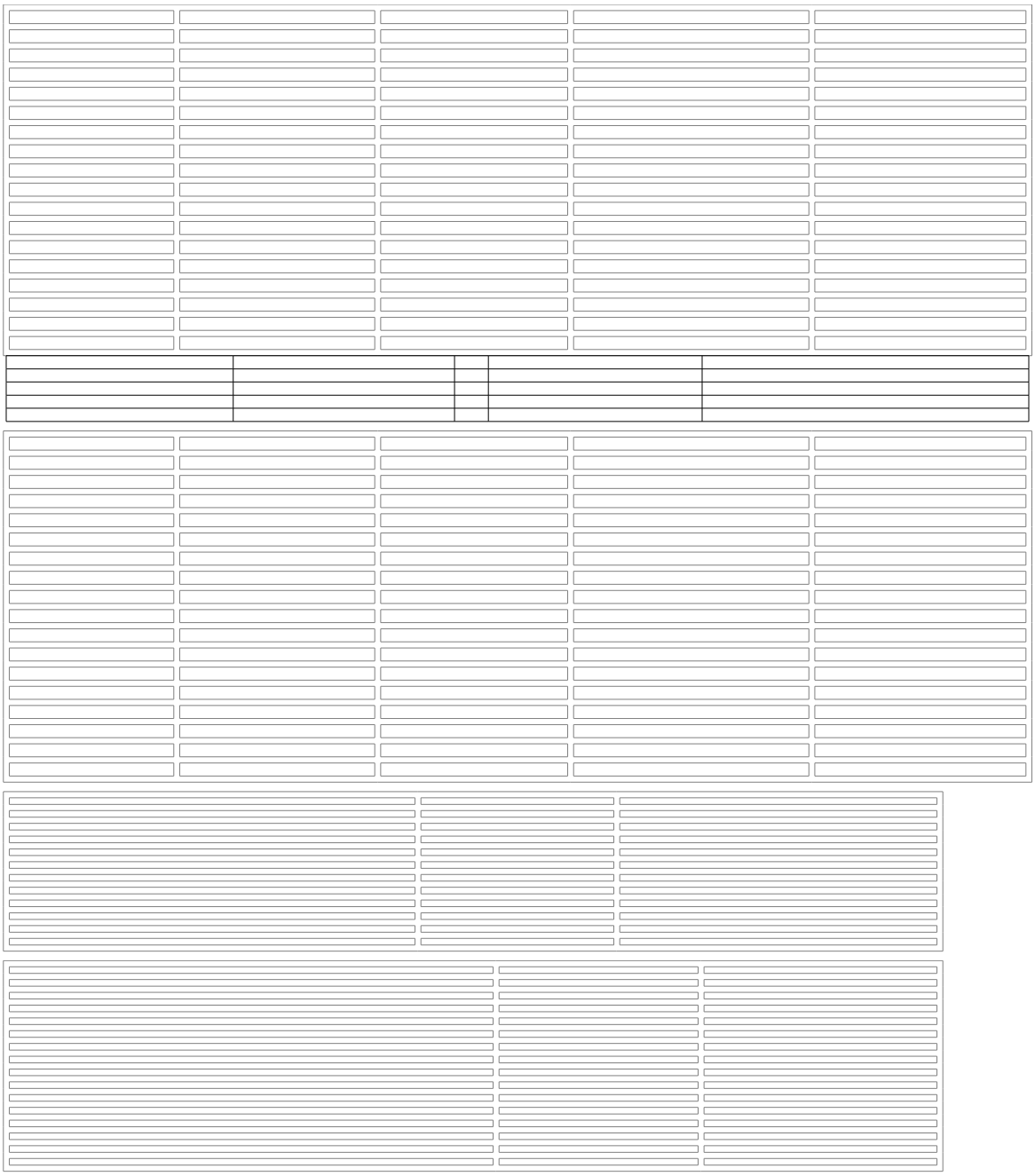
Note: This function uses linear regression. How can we codify the specification to be actually useful without including the whole theory behind it? For the moment, we won't give the details; they are lengthy, and in any case well-known. Simply stating that it is a "linear regression" should be sufficient for the purposes of this spec.

TODO: Test case.



Copyright © OASIS Open 2006. All Rights Reserved.

Copyright © OASIS Open 2006. All Rights Reserved.



6.17.44 MAX

Summary: Return the maximum from a set of numbers.

Syntax: MAX({ *NumberSequenceList* N }$^{+}$)

Returns: Number

Constraints: None.

Copyright © OASIS Open 2006. All Rights Reserved.

Semantics: Returns the value of the maximum number in the list passed in. Non-numbers are ignored. Note that if logical types are a distinct type, they are not included. What happens when MAX is provided 0 parameters is implementation-defined, but MAX with no parameters **should** return 0.

Test Cases:

Expression	*Result*	*Comment*
=MAX(2;4;1;-8)	4	Negative numbers are smaller than positive numbers.
=MAX([.B4:.B5])	3	The maximum of (2,3) is 3.
=ISNA(MAX(NA()))	True	Inline errors *are* propagated.
=MAX([.B3:.B5])	3	Strings are not converted to numbers and are ignored.
=MAX(-1;[.B7])	-1	Strings are not converted to numbers and are ignored.
=MAX([.B3:.B9])	Error	Errors inside ranges are NOT ignored.

See also MAXA, MIN

Note: There was debate on whether or not we should we mandate support for zero parameters. Excel 2002 and OOo don't support it. Gnumeric does. The compromise above gives a recommendation without mandating it.

6.17.45 MAXA

6.17.46 MEDIAN

Summary: Returns the median (middle) value in the list.

Syntax: MEDIAN({ *NumberSequenceList* X}+)

Returns: Number

Semantics:

MEDIAN logically ranks the numbers (lowest to highest). If given an odd number of values, MEDIAN returns the middle value. If given an even number of values, MEDIAN returns the arithmetic average of the two middle values.

Copyright © OASIS Open 2006. All Rights Reserved.

$$n = is\,the\,count\,of\,the\,ranked\,numbersequence$$

$$\tilde{x} = x_{\left(\frac{(n+1)}{2}\right)}$$
$$for\,n = odd$$

$$\tilde{x} = \frac{1}{2}\left(x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}\right)$$
$$for\,n = even$$

Test Cases:

Expression	*Result*	*Comment*
=MEDIAN([.E51:.E52])	8.850000000±ε	
=MEDIAN([.E52;.E54];45)	7.200000000±ε	
=MEDIAN([.E52:.E55])	7.650000000±ε	
=MEDIAN(1;3;13;14;15)	13.000000000±ε	
=MEDIAN(1;3;13;14;15;35)	13.500000000±ε	

6.17.47 MIN

Summary: Return the minimum from a set of numbers.

Syntax: MIN({ *NumberSequenceList* N }$^{+}$)

Returns: Number

Constraints: None.

Semantics: Returns the value of the minimum number in the list passed in. Returns zero if no numbers are provided in the list. What happens when MIN is provided 0 parameters is implementation-defined, but MIN() with no parameters **should** return 0.

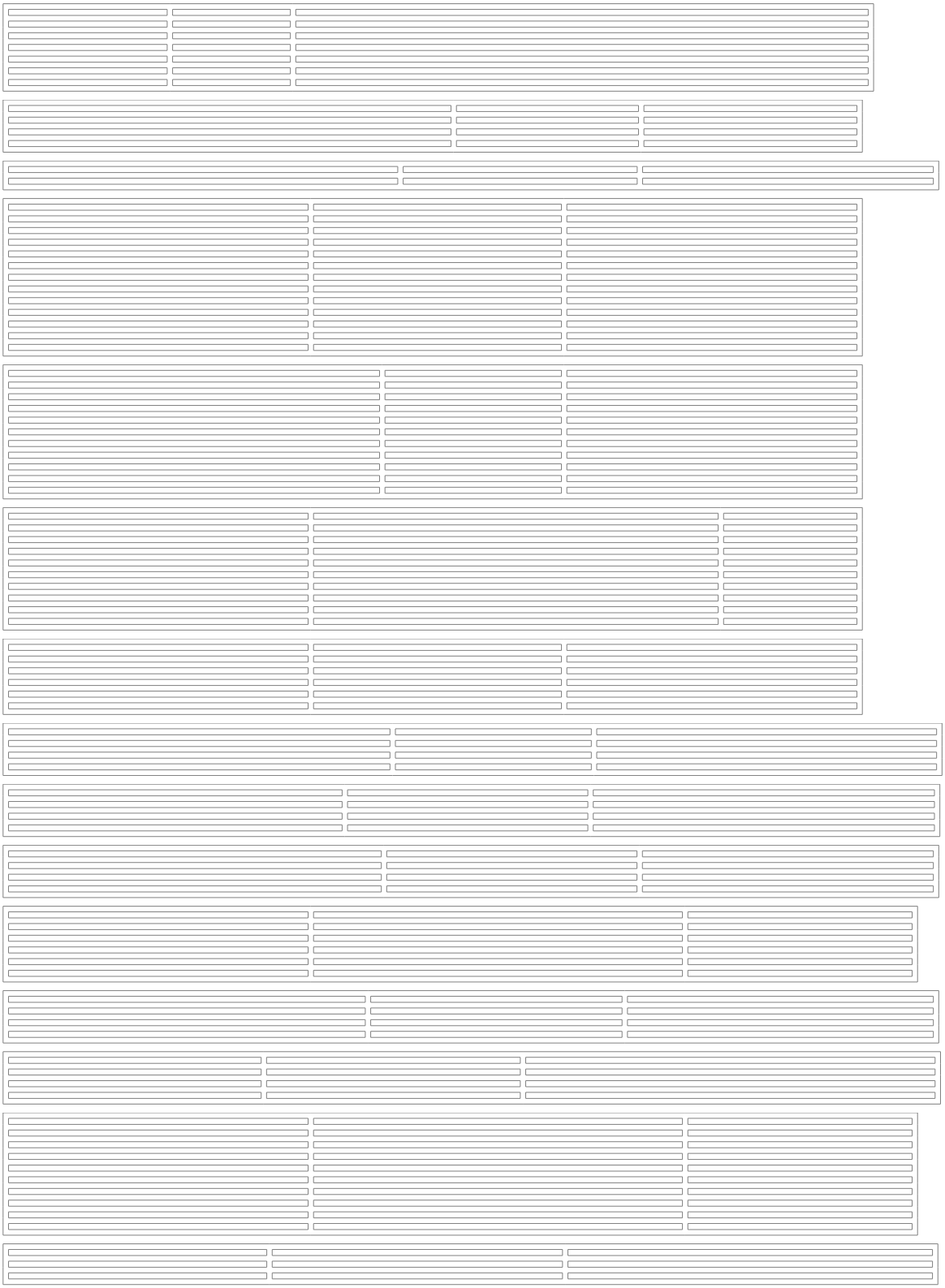
Test Cases:

Expression	*Result*	*Comment*
=MIN(2;4;1;-8)	-8	Negative numbers are smaller than positive numbers.
=MIN([.B4:.B5])	2	The minimum of (2,3) is 2.
=MIN([.B3])	0	If no numbers are provided in all ranges, MIN returns 0
=MIN("a")	Error	Non-numbers inline are NOT ignored.
=MIN([.B3:.B5])	2	Cell text is not converted to numbers and is ignored.

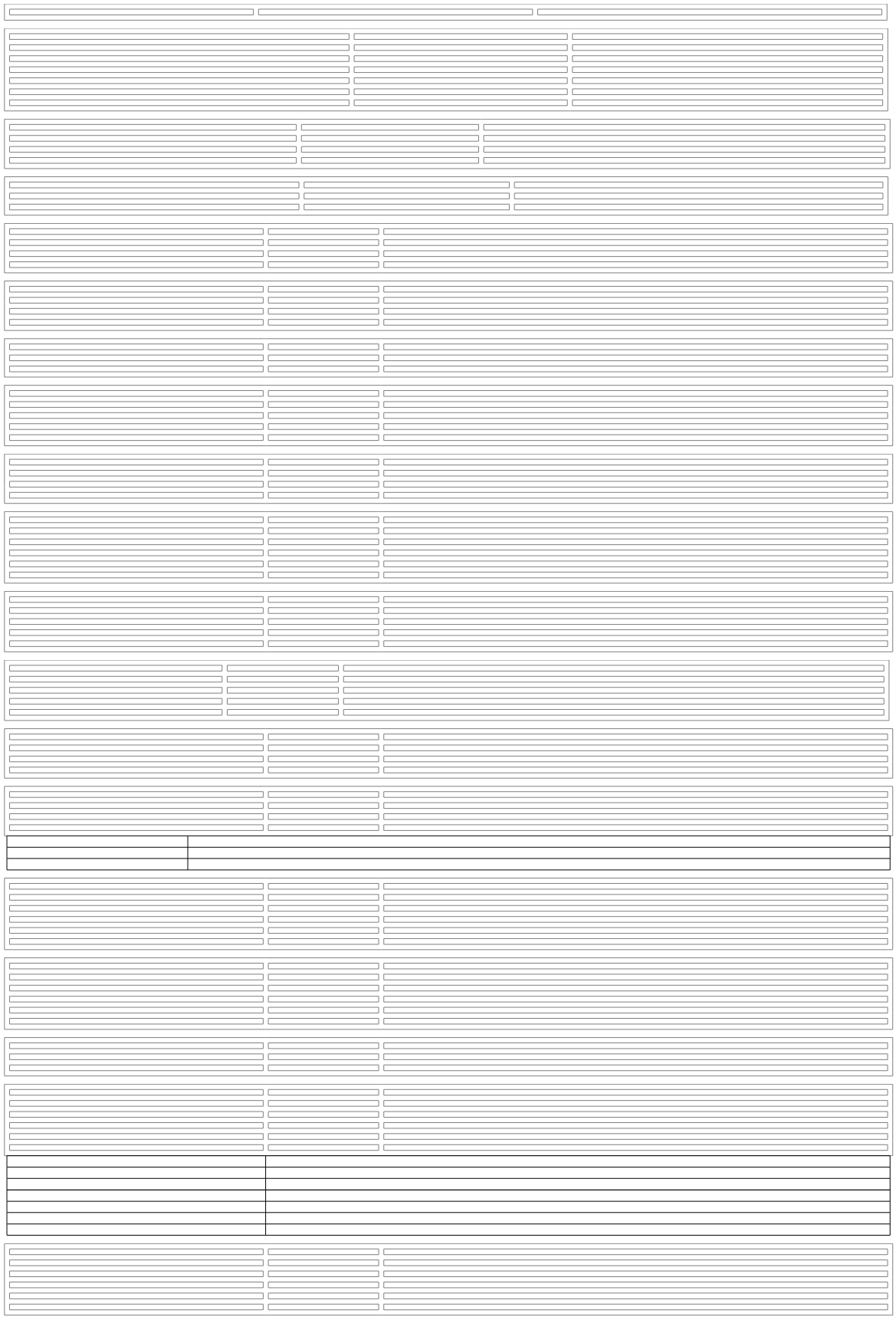
Copyright © OASIS Open 2006. All Rights Reserved.

Note: There was debate on whether or not we should we mandate support for zero parameters; see the discussion for MAX.

See also MAX, MINA



Copyright © OASIS Open 2006. All Rights Reserved.



Copyright © OASIS Open 2006. All Rights Reserved.

6.17.81 VAR

Summary: Compute the sample variance of a set of numbers.

Syntax: VAR({ *NumberSequence* N }$^{+}$)

Returns: Number

Constraints: At least two numbers must be included. Returns an error if less than two numbers are provided.

Semantics: Computes the sample variance s^2, where

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2 = \frac{1}{n-1}\left(\left(\sum_{i=1}^{n} x_i^{\,2}\right) - n\,\overline{x}^2\right).$$

Note that s^2 is not the same as the variance of the set, σ^2, which uses n rather than $n - 1$.

Test Cases:

Expression	*Result*	*Comment*
=VAR(2;4)	2	The sample variance of (2;4) is 2.
=VAR([.B4:.B5])*2	1	The sample variance of (2;3) is 0.5.
=VAR([.B3:.B5])*2	1	Strings are not converted to numbers and are ignored.
=VAR(1)	Error	At least two numbers must be included

See also VARP, STDEV, AVERAGE

TBD: Should we specify what happens with non-numbers? In Excel 2002, it produces an error.

6.17.82 VARA

Summary: Estimates the variance using a sample set of numbers.

Syntax: VARA({ *Any* sample }$^{+}$)

Returns: Number

Constraints: The sequence must contain two numbers at least.

Semantics: Estimates the variance using a sample set of numbers.

Given the expectation value $\overline{x}$ the estimated variance becomes

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2 = \frac{1}{n-1}\left(\left(\sum_{i=1}^{n} x_i^{\,2}\right) - n\,\overline{x}^2\right)$$

In the sequence, only numbers and logical types are considered; cells with text are converted to 0; other types are ignored. If logical types are a distinct type, they are still included, with True considered 1 and False considered 0. Any *sample* may be of type ReferenceList.

Copyright © OASIS Open 2006. All Rights Reserved.

The handling of strings is implementation defined. Either, strings are converted to numbers, if possible and otherwise, they are treated as zero, or strings are always treated as zero.

Test Cases:

Expression	*Result*	*Comment*
=VARA(2;4)	2	The sample variance of (2;4) is 2.
=VARA([.B5:.C6])	6.666667±ε	Logicals (referenced) are converted to numbers.
=VARA(TRUE(); FALSE())	1	Logicals (inlined) are converted to numbers.
=VARA(1)	Error	Two numbers at least.

See also VAR

Note:
OOo Calc (2.02): Treats strings as zero in VARA, even if they would be convertible. Ignores them for VAR. Converts Logicals to {0,1} in both cases, VAR and VARA.
Gnumeric (1.6.1): Converts logicals and strings (if possible) for VARA. Ignores both in VAR.

6.17.83 VARP

Summary: Compute the variance of the set for a set of numbers.

Syntax: VARP({ *NumberSequence* N }$^{+}$)

Returns: Number

Constraints: At least two numbers must be included; it is unspecified what is returned with only one number. Returns an error if less than two numbers are provided.

Semantics: Computes the variance of the set σ^2, where

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2 = \frac{1}{n}\left(\left(\sum_{i=1}^{n} x_i^{\,2}\right) - n\,\overline{x}^2\right).$$

Note that σ^2 is not the same as the sample variance, s^2, which uses $n - 1$ rather than n.

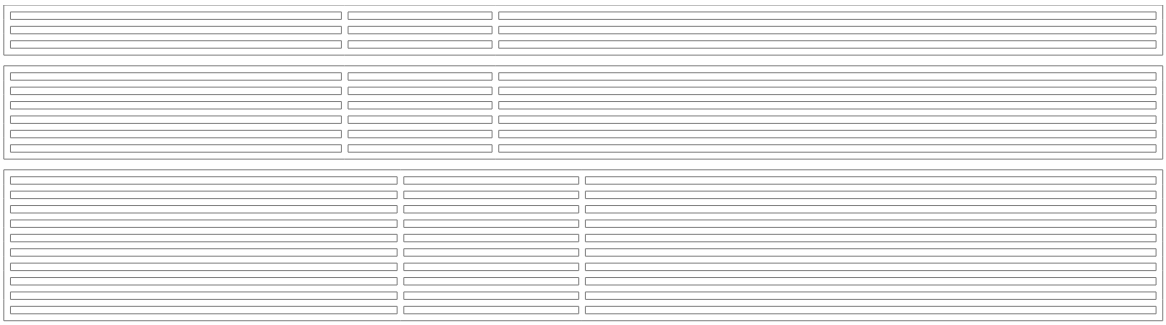
Test Cases:

Expression	*Result*	*Comment*
=VARP(2;4)	1	The variance of the set for (2;4) is 1.
=VARP([.B4:.B5])*4	1	The variance of the set for (2;3) is 0.25.
=VARP([.B3:.B5])*4	1	Strings are not converted to numbers and are ignored.

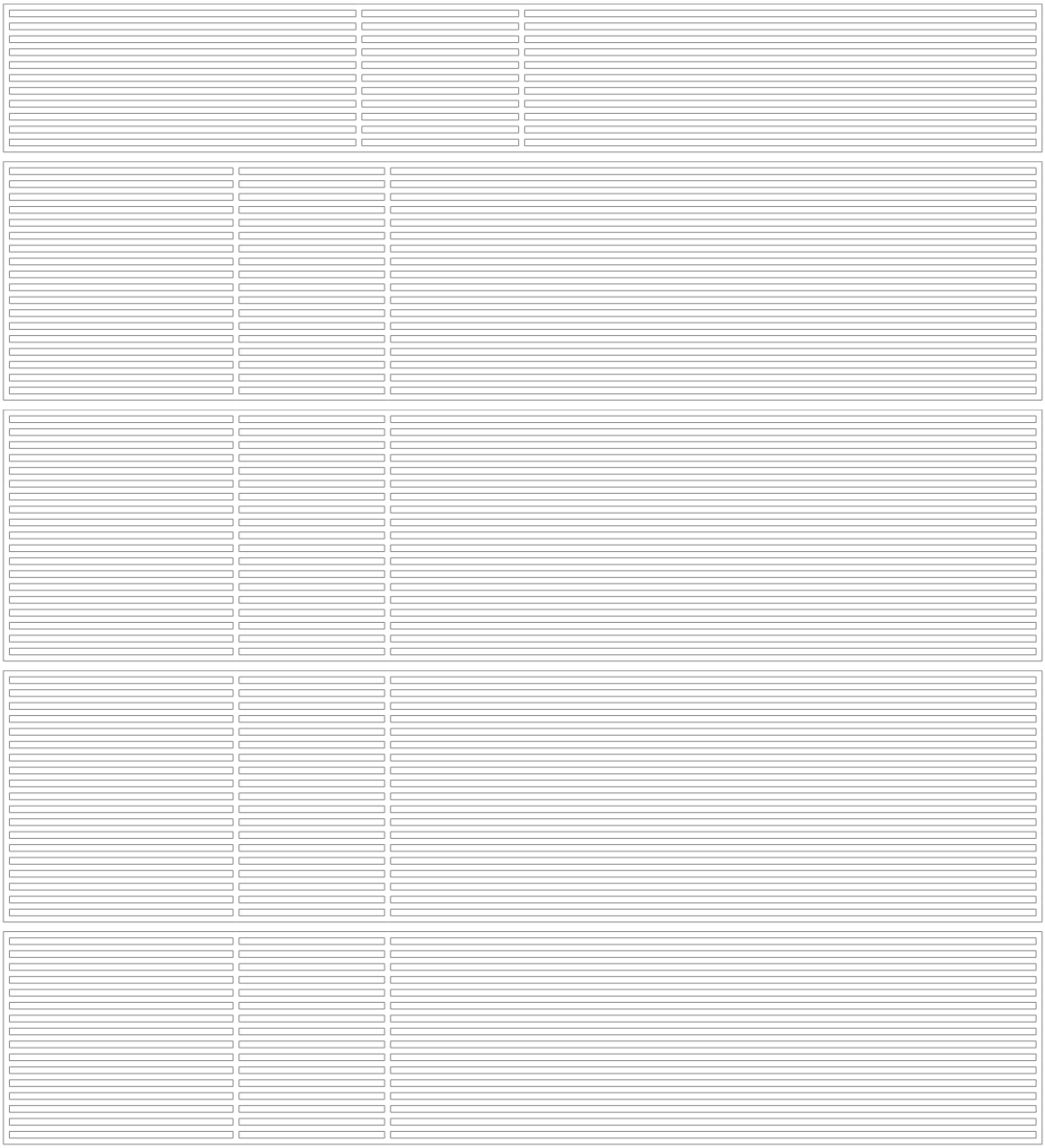
TBD: VARP(1) shouldn't be required to be an error. Excel, at least, accepts it.

See also VAR, STDEVP, AVERAGE

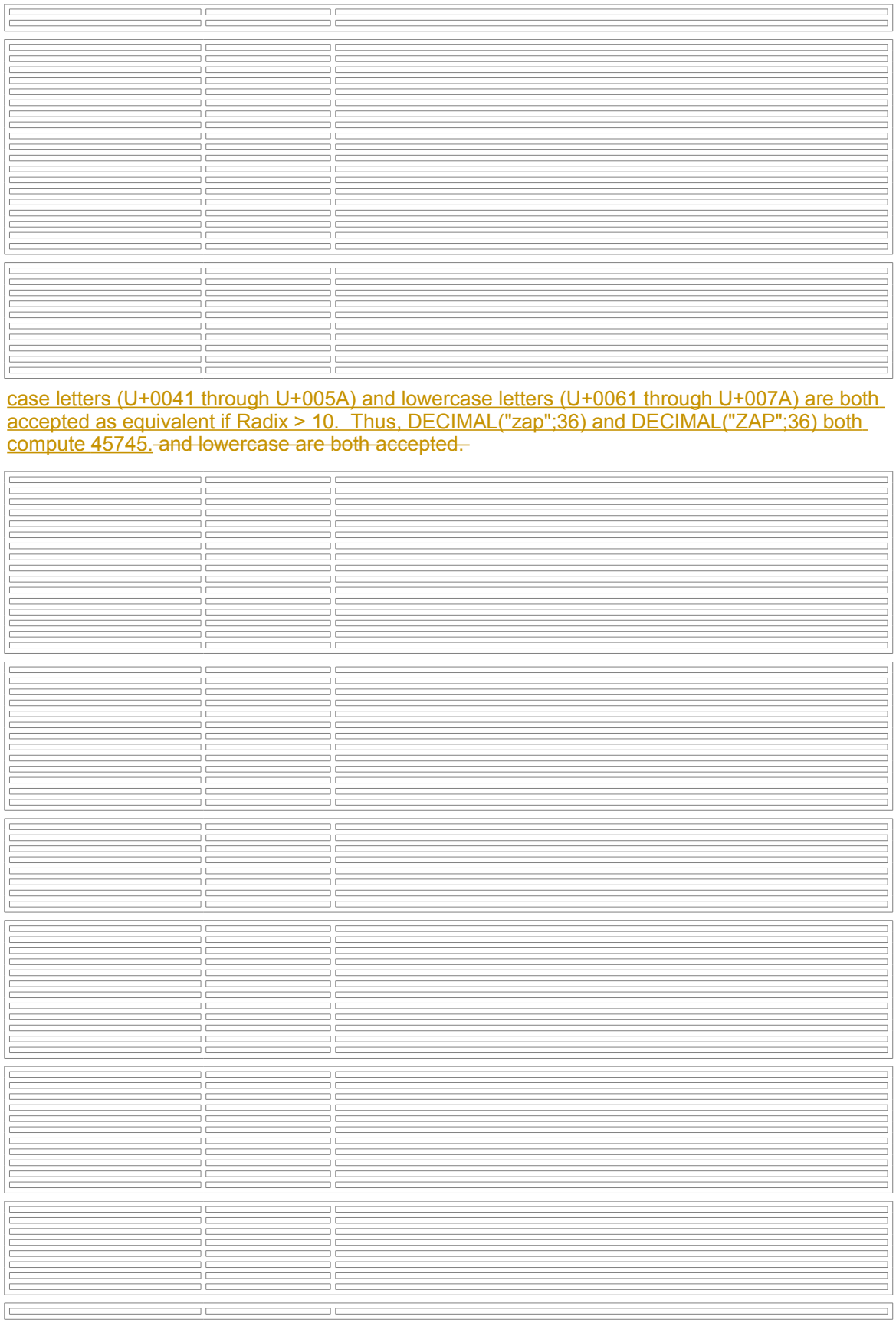
Copyright © OASIS Open 2006. All Rights Reserved.



that represents the value of X in base Radix. The symbols 0-9 (U+0030 through U+0039), then upper case A-Z (U+0041 through U+005A) are used as digits. Thus, BASE(45745;36) returns "ZAP"~~of base Radix to a Number. The symbols 0-9, then A-Z (upper case) are used~~

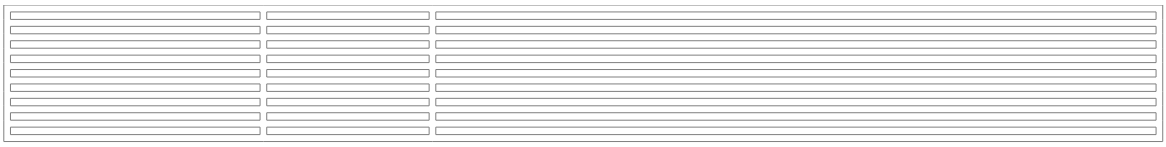


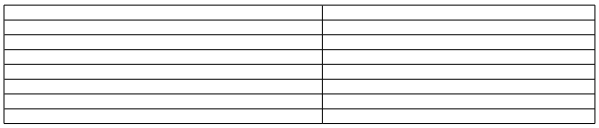
Copyright © OASIS Open 2006. All Rights Reserved.

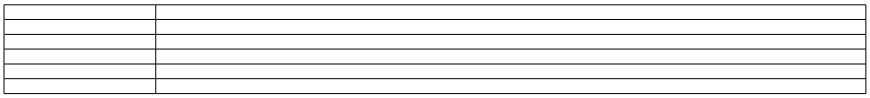


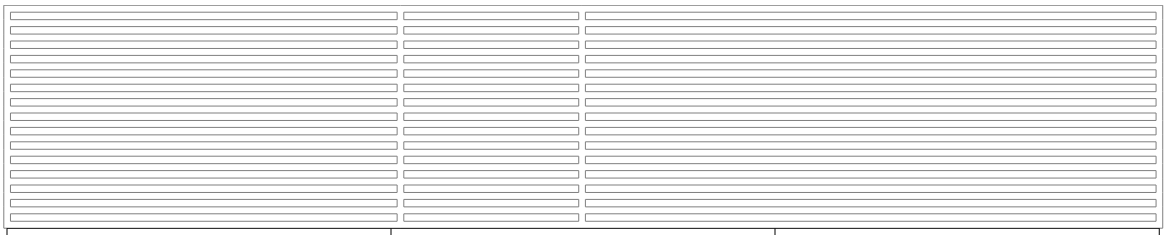
case letters (U+0041 through U+005A) and lowercase letters (U+0061 through U+007A) are both accepted as equivalent if Radix > 10. Thus, DECIMAL("zap";36) and DECIMAL("ZAP";36) both compute 45745. ~~and lowercase are both accepted.~~

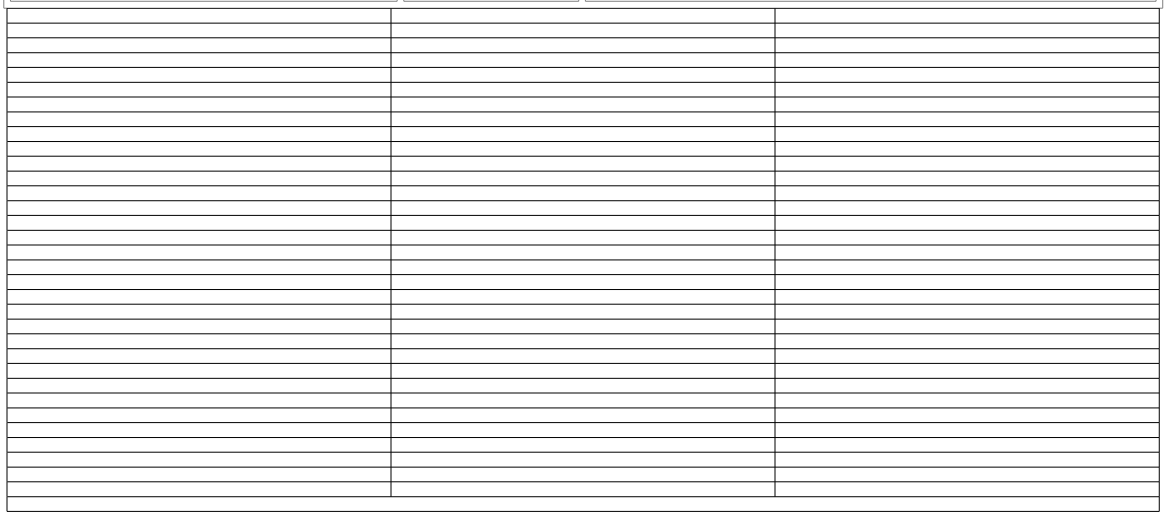
Copyright © OASIS Open 2006. All Rights Reserved.

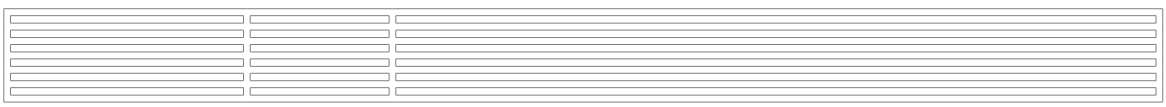




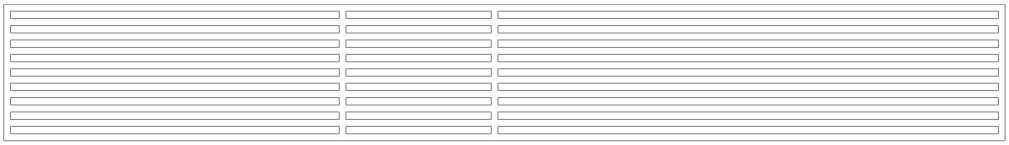
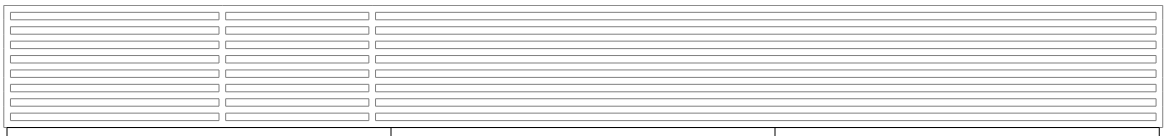
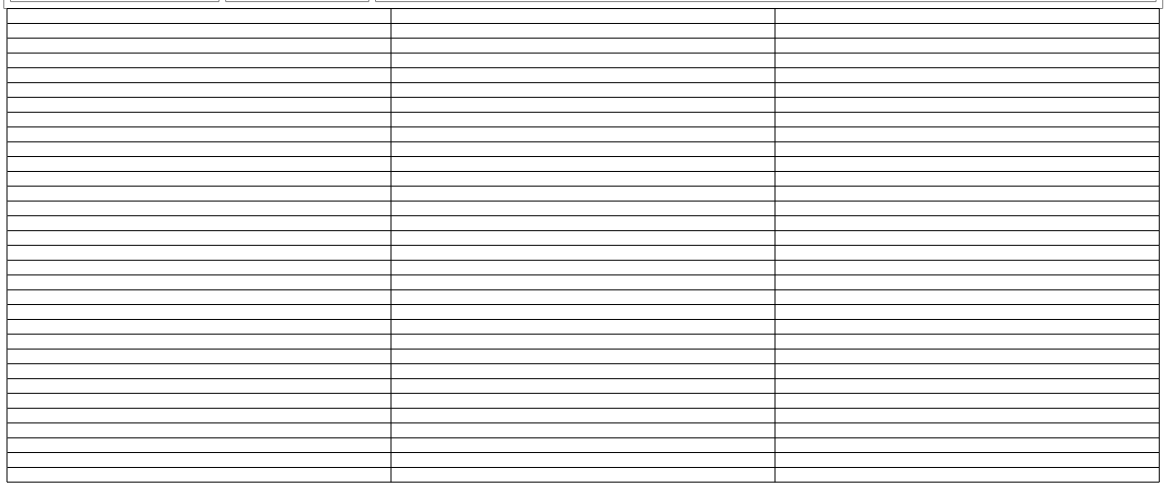


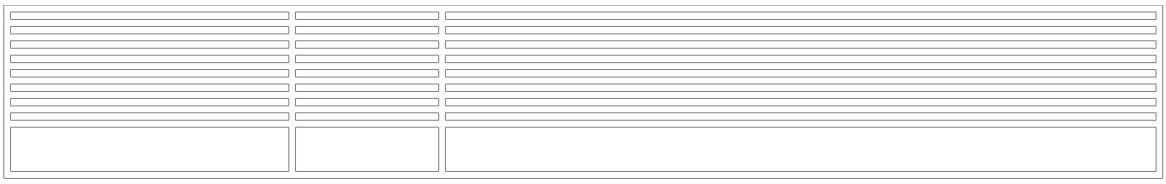
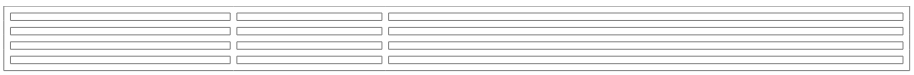


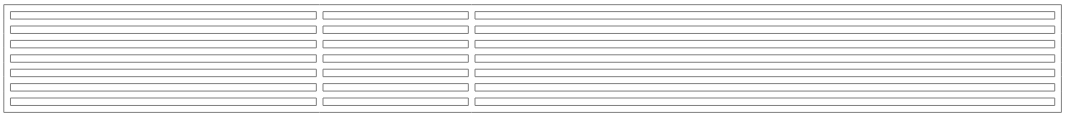


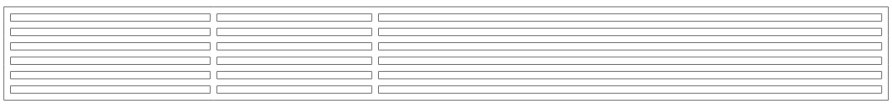
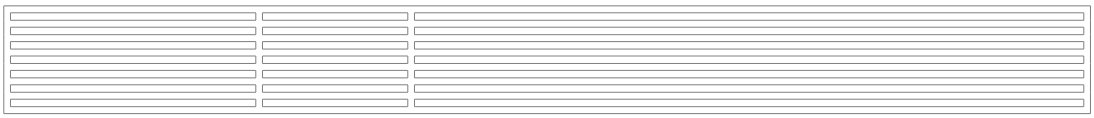


Copyright © OASIS Open 2006. All Rights Reserved.

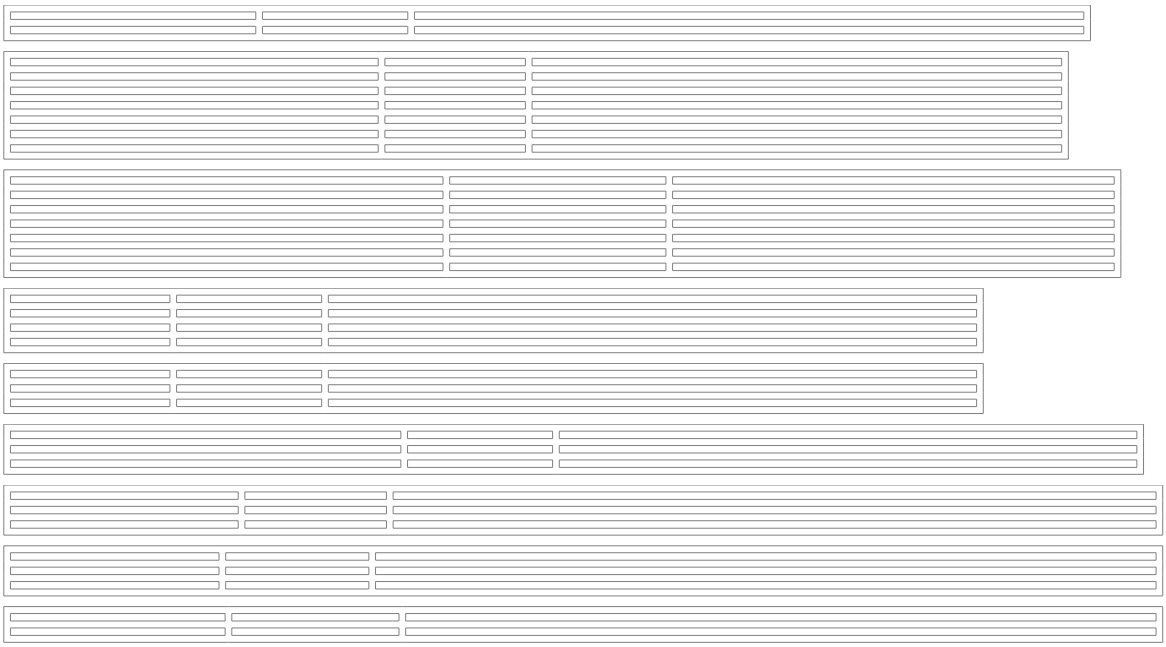







Copyright © OASIS Open 2006. All Rights Reserved.



Copyright © OASIS Open 2006. All Rights Reserved.

7 Other Capabilities

Applications **may** implement additional abilities that are not simply a matter of which function they support. The following sections describe some specific additional capabilities; applications **may** implement them, and documents **may** require them (though such documents may not be correctly recalculated on applications which do not implement them). Documents that depend on these other capabilities can still be considered "portable documents", but only if these additional capabilities are clearly noted (since not applications implement these additional capabilities).

TODO: Place test cases for arrays here. Note that the definition of array *processing* goes earlier, in the processing model section, along with a note that not all implementations support arrays.

7.1 Inline constant arrays

Applications claiming to implement "Inline constant arrays" **shall** support inline arrays with one matrix, with one or more rows, and one or more columns. Such applications **shall** support these 2-dimensional arrays as long as the number of expressions in each row is identical; applications **may** but need not support arrays with a different number of expressions in each row. They **shall** support *at least* the following syntactic rules in the Expression values for the inline array:

- Number, optionally preceded with the prefix "-" operator (for negative numbers)
- Text
- Logical constants TRUE() and FALSE()
- Error

Test Cases:

Expression	***Result***	***Comment***
=LARGE({1;2;3};1)	3	
=LARGE({1;2;3};3)	1	
=LARGE({1;2;3};4)	Error	N is greater than the length of the list
=SMALL({1;2;3};1)	1	
=SMALL({1;2;3};3)	3	
=SMALL({1;2;3};4)	Error	N is greater than the length of the list
=CORREL({1;2;3;4};{1;2;3;4})	1	Perfect positive correlation
=CORREL({1;2;3;4};{4;3;2;1})	-1	Perfect negative correlation
=CORREL({1;2;3};{2;3})	Error	The length of each list must be equal

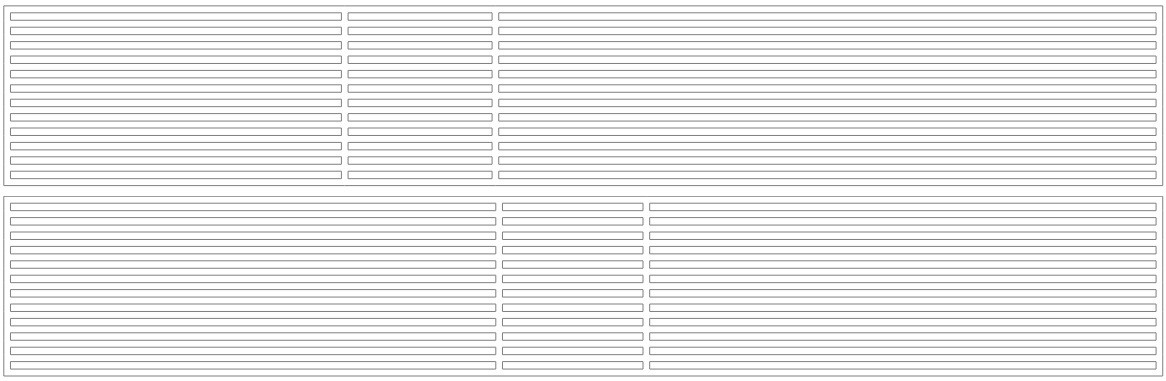
TODO: Add tests for multirow, multicolumn, etc.

Copyright © OASIS Open 2006. All Rights Reserved.

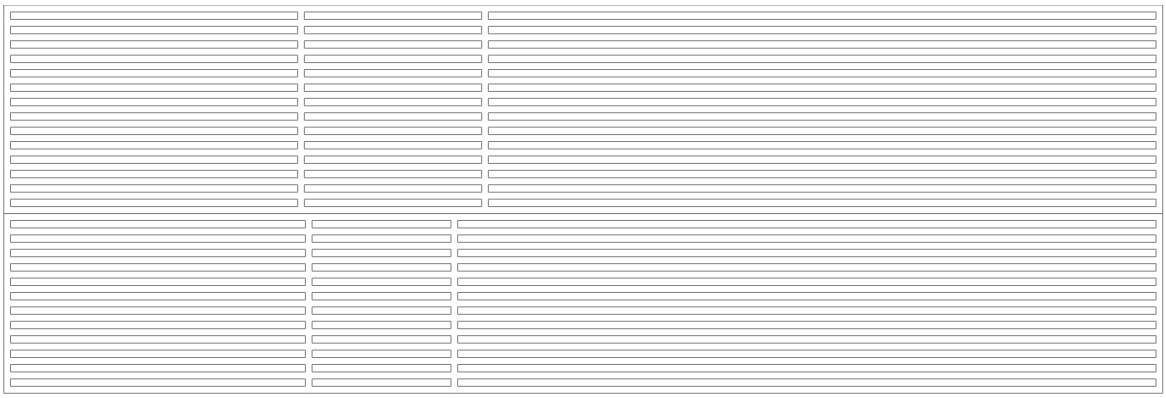
7.2 Inline non-constant arrays

Applications claiming to implement “Inline non-constant arrays” **shall** support the full Expression syntax in each component of an array (and not just constants).

TODO: Tests



Copyright © OASIS Open 2006. All Rights Reserved.



Copyright © OASIS Open 2006. All Rights Reserved.

Appendix A.References

[ISO10646] ISO. ISO/IEC 10646:2003. Information technology — Universal Multiple-Octet Coded Character Set (UCS). 2003.

[XML1.1] W3C. Extensible Markup Language (XML) 1.1 (Second Edition). W3C Recommendation 16 August 2006 http://www.w3.org/TR/2006/REC-xml11-20060816/

[Zwillinger] Zwillinger, Daniel (editor-in-chief). CRC Standard Mathematical Tables and Formulae. Chapman & Hall/CRC, CRC Press Company. 31st edition. 2003.

[CSS2] Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs, *Cascading Style Sheets, level 2*, http://www.w3.org/TR/1998/REC-CSS2-19980512, W3C, 1998.

[CSS3Text] Michel Suignard, *CSS3 Text Module*, http://www.w3.org/TR/2003/CR-css3-text-20030514, W3C, 2003.

[DAISY] *ANSI/NISO Z39.86-2005 Specifications for the Digital Talking Book*, http://www.niso.org/standards/resources/Z39-86-2005.html, 2005

[DCMI] -, *Dublin Core Metadata Element Set, Version 1.1: Reference Description*, http://www.dublincore.org/documents/dces/, Dublin Core Metadata Initiative, 2003.

[DOM2] W3C, *Document Object Model Level 2 Core Specification*, http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113, W3C, 2000.

[DOMEvents2] Tom Pixley, *Document Object Model (DOM) Level 2 Events Specification*, http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113, W3C, 2000.

[DOMEvents3] Philippe Le Hégaret, Tom Pixley, *Document Object Model (DOM) Level 3 Events Specification*, http://www.w3.org/TR/DOM-Level-3-Events/, W3C, 2003.

[HTML4] Dave Raggett, Arnoud Le Hors, Ian Jacobs, *HTML 4.01 Specification*, http://www.w3.org/TR/1999/REC-html401-19991224, W3C, 1999.

[ISO/IEC Directives] ISO/IEC Directives, Part 2 *Rules for the structure and drafting of International Standards*, 2004

[JDBC] Jon Ellis, Linda Ho, Maydene Fisher, *JDBC 3.0 Specification*, http://java.sun.com/products/jdbc/, Sun Microsystems, Inc., 2001.

[MathML] David Carlisle, Patrick Ion, Robert Miner, Nico Poppelier, *Mathematical Markup Language (MathML) Version 2.0 (Second Edition)*, http://www.w3.org/TR/2003/REC-MathML2-20031021/, W3C, 2003.

[MIMETYPES] , *List of registered MIME types*, ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/, IANA, .

[OLE] Kraig Brockschmidt, Inside OLE, Microsoft Press, 1995, ISBN: 1-55615-843-2

[OOo] , *OpenOffice.org XML File Format 1.0 Technical Reference Manual*, http://xml.openoffice.org/xml_specification.pdf, Sun Microsystems, Inc., 2002.

[PNG] Thomas Boutell, *PNG (Portable Network Graphics) Specification*, http://www.w3.org/TR/REC-png-multi.html, W3C, 1996.

[RFC1766] H. Alvestrand, *Tags for the Identification of Languages*, http://www.ietf.org/rfc/rfc1766.txt, IETF, 1995.

Copyright © OASIS Open 2006. All Rights Reserved.

[RFC2045] N. Freed and N. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, http://www.ietf.org/rfc/rfc2045.txt, IETF, 1996.

[RFC2048] N. Freed, J. Klensin, J. Postel, *Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures*, http://www.ietf.org/rfc/rfc2048.txt, IETF, 1996.

[RFC2616] IETF, *Hypertext Transfer Protocol -- HTTP/1.1*, http://www.ietf.org/rfc/rfc2616.txt, IETF, 1999.

[RFC2898] B. Kaliski, *PKCS #5: Password-Based Cryptography Specification Version 2.0*, http://www.ietf.org/rfc/rfc2898, IETF, 2000.

[RFC3066] H. Alvestrand, *Tags for the Identification of Languages*, http://www.ietf.org/rfc/rfc3066.txt, IETF, 2001.

[RFC3987] M. Duerst, M. Suignard, *Internationalized Resource Identifiers (IRIs)*, http://www.ietf.org/rfc/rfc3987.txt, IETF, 2005.

[RNG] ISO/IEC 19757-2 *Document Schema Definition Language (DSDL) -- Part 2: Regular-grammar-based validation -- RELAX NG*, 2003

[RNG-Compat] James Clark, MURATA Makoto, *RELAX NG DTD Compatibility*, http://www.oasis-open.org/committees/relax-ng/compatibility-20011203.html, OASIS, 2001.

[SMIL20] W3C, *Synchronized Multimedia Integration Language 2.0 (SMIL 2.0)*, http://www.w3.org/TR/smil20/, W3C, 2001.

[SVG] Jon Ferraiolo, 藤沢 淳 (FUJISAWA Jun), Dean Jackson, *Scalable Vector Graphics (SVG) 1.1*, http://www.w3.org/TR/2003/REC-SVG11-20030114/, W3C, 2003.

[UAX9] Mark Davis, Unicode Standard Annex #9: *The Bidirectional Algorithm, Version 15 or later*, http://www.unicode.org/reports/tr9/tr9-15.html, 2005

[UNICODE] The Unicode Consortium. The Unicode Standard, Version 4.0.0, defined by: *The Unicode Standard, Version 4.0* (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1)

[UTR20] Martin Dürst and Asmus Freytag, Unicode Technical Report #20: *Unicode in XML and other Markup Languages*, http://www.unicode.org/reports/tr20/, 2003

[XForms] W3C, *XForms*, http://www.w3.org/TR/xforms/, W3C, 2004.

[XLink] Steve DeRose, Eve Maler, David Orchard, *XML Linking Language*, http://www.w3c.org/TR/xlink/, W3C, 2001.

[xml-names] Tim Bray, Dave Hollander, Andrew Layman, *Namespaces in XML*, http://www.w3.org/TR/REC-xml-names/, W3C, 1999.

[XML1.0] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau , *Extensible Markup Language (XML) 1.0 (Third Edition)*, http://www.w3.org/TR/2004/REC-xml-20040204, W3C, 2004.

[xmlschema-2] Paul V. Biron, Ashok Malhotra, *XML Schema Part 2: Datatypes Second Edition*, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/, W3C, 2004.

[xmlschema-2] Paul V. Biron, Ashok Malhotra, *XML Schema Part 2: Datatypes*, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/, W3C, 2001.

[XSL] W3C, *Extensible Stylesheet Language (XSL)*, http://www.w3.org/TR/2001/REC-xsl-20011015/, W3C, 2001.

[XSLT] James Clark, *XSL Transformations (XSLT) Version 1.0*, http://www.w3.org/TR/1999/REC-xslt-19991116, W3C, 1999.

Copyright © OASIS Open 2006. All Rights Reserved.

[XSLT2] Michael Kay, *XSL Transformations (XSLT) Version 2.0*, http://www.w3.org/TR/2003/WD-xslt20-20031112/, W3C, 2003.

[ZIP] *Info-ZIP Application Note 970311*, ftp://ftp.uu.net/pub/archiving/zip/doc/appnote-970311-iz.zip, 1997

TODO: Fix formatting.

Additional references:

- Blattner, Patrick, Laurie, Ulrich, Ken Cook, and Timothy Dyck (1999). *Special Edition Using Microsoft Excel*. Que. Indianapolis, Indiana. ISBN 0-7897-1729-8.
- Simon, Jinjer (2000). *Excel 2000 in a Nutshell*. O'Reilly.
- Walden, Jeff (1986). *File Formats for Popular PC Software: A Programmer's Reference*. Wiley Press Book, John Wiley & Sons, Inc. NY, NY.
- Walkenbach, John (1999). *Excel 2000 Formulas*. M&T Books, an imprint of IDG Books Worldwide, Inc. ISBN 0-7645-4609-0.
- Walkenbach, John (2004). *Excel 2003 Formulas*. Wiley Publishing, Inc. Indianapolis, Indiana. ISBN 0-7645-4073-4.

Copyright © OASIS Open 2006. All Rights Reserved.

Appendix B.Acknowledgments (Non Normative)

The effort to develop this formula specification was led by David A. Wheeler (office formula subcommittee chair), Eike Rathke, and Robert Weir (office formula subcommittee editors). We gratefully acknowledge the contributions of all participants in its development:

Waldo Bastian
Casper Boemann
Nathaniel Borenstein
Daniel Bricklin
Daniel Carrera
John Cowan
Gary Edwards
J. David Eisenberg
Jason Faulkner
David Faure
Jody Goldberg
Andreas J. Guelzow
Dennis E. Hamilton
Ariya Hidayat
Richard Kernick
Tomas Mecir
Thomas Metcalf
Tom Metcalf
Laurent Montel
Adam Moore
Niklas Nebel
Eike Rathke
Richard Rothwell
Herbert Schiemann
Jon Smirl
Frank Stecher
James Richard Tyrer
Inge Wallin
Robert Weir
Morten Welinder
David A. Wheeler
Kohei Yoshida
Helen Yue

Copyright © OASIS Open 2006. All Rights Reserved.

Appendix C.Index

TODO: Insert an index here, at least one entry for every function, type, and syntactic rule.

Copyright © OASIS Open 2006. All Rights Reserved.