Spreadsheet Data Description Language

Introduction

This is the definition of our prototype spreadsheet description language. It defines domain specific concepts, a rich type system and data extraction directives that distinguish it from existing metadata languages such as CVS On The Web and CsvSchema.

Structure

The low-level language is structured as a series of directives and pragmas that appear in a stream in a linear fashion. There are no loops or nesting. Directives are used to describe the structure of a particular spreadsheet. Pragmas are used to set and configure language level options.
Each directive or pragma consists of a verb, followed by a number of arguments.
We define a simple serialisation of the language that is similar in structure to a TSV (tab separated value) file. This allows it to be edited and manipulated with a regular spreadsheet program. We also define an embedding of this language that can be placed as the first sheet of an Excel or OpenDocument Workbook. This allows a spreadsheet and its description to be self-contained.
There are also some other useful serialisations of this language. For example, consuming parties may wish to publish a metadata file that contains a high level description of what they would like to receive. Publishing parties may wish to augment this high level description with an additional metadata file that specified a lower level description of their spreadsheet and "joins it up" with the higher level metadata.
The language defined herein has the concept of data types. These types can be low level things that denote the serialisation and/or semantics of something like a "number" an "integer" or a "string". They can also be higher level types such as "currency", "count of items" or even domain specific types such as "GBP including VAT @ 20%" or "number of laptops". This standard defines the lower level types directly along with a method of declaring higher level types as well as a way of publishing, sharing and managing these definitions securely in a distributed, multi-stakeholder ecosystem.
Finally, we define some directives that allow users to control permission, security and disclosure.

Types

Specified the kind of data to try to extract from a cell. If the cell contains a formula but the type denotes a non-formula type then the implementation should attempt to extract some data of the declared type from the cached evaluation. If the cached data is not available then

the extraction MUST fail. If the cached data is available then the implementation MAY emit a warning.

Built-in Types

String

Number

Formula

User Defined Types

GBPxVAT

Directives

These are the things that appear inside a spreadsheet description language command stream. Such a stream consists of a sequence of ordered directives and each directive appears along with its arguments as defined below.
Directives and their arguments can be any UTF-8 codepoint. The contents of arguments can be further restricted by the definition for each directive. Each serialisation format will define how the codepoints should be escaped or encoded.

declare-header

This directive declares where the header of the tabular data contained within the spreadsheet can be found. It MUST be a single row or column of cells. The `declare-data` directive is used in conjunction with this directive to determine whether the data is formated as a series of columns or a series of rows.

`declare-header <range>`

`<range>`	Either a range of cells denoted using the standard Excel range format (i.e `A1:B2`) or a named range that can be expected to be declared by the spreadsheet data file (i.e `Sheet1!NamedRange`). A range MUST describe a single column or a single row. The degenerate case of a single cell is accepted.

This directive MUST appear exactly once in each metadata file.
This directive MUST appear exactly once when multiple metadata files are combined for processing a single spreadsheet.

Limitations

1. This version of the language standard does not support extracting data from pivot tables. This functionality will be handled in a future revision of the language. "Pivot tables" are considered as any table that contains hierarchical column or row headings as well as tables that require both column and row header declarations.

declare-type

This directive maps a heading name that appears inside a spreadsheet header to a type that is used to validate and deserialise the data from that column or row in the data section.
`declare-type "<header-name>" <type>`

`<header-name>`	The contents of the cell that contains the heading. As the cell contents is contained within double quotes, double quotes that occur within the cell contents must be escaped with a BACKSLASH character. NOTE: The Tab Separated Value Serialisation of this language also requires its own escaping around BACKSLASH characters. Therefore, to embed a double quote in a `<header-name>`, it should be encoded as `\"` and to embed that in a Tab Separated Value Serialisation, it should be encoded as `\\"`.
`<type>`	Type type of the data in the denoted column or row. This type should be one of the types defined below or a user defined type from an available registry.

This directive MAY appear any number of times in each metadata file however, it MUST NOT appear more than once for a given variant of `<header-name>`.
This directive MAY appear any number of times when multiple metadata files are combined for processing a single spreadsheet however, it MUST NOT appear more than once for a given variant of `<header-name>`.

Limitations

1. The `header-name` MUST be a string that is equal to the contents of the cell that contains the heading. Where the cell type of the relevant heading cell is not encoded as a string in the spreadsheet, the behaviour of this directive is undefined.
2. We do not currently support anything other than matching against the cell contents. Therefore we cannot handle spreadsheets that contain distinct columns with the same heading. Conformant implementations MUST emit a warning when this is detected but SHOULD proceed with the processing anyway.
3. We do not currently support declaring a type through absolute indexing into the header array. This would allow unambiguous column type declarations and would also enforce a specific column ordering.

declare-data

This directive denotes where the tabular data appears inside a workbook.
This directive MUST appear exactly once in each metadata file.
This directive MUST appear exactly once when multiple metadata files are combined for processing a single spreadsheet.

#

This directive denotes a comment and should be ignored. It can have any number (including zero) of arguments.

```
# This is a comment and should be ignored.
```

This directive MAY appear any number of times in each metadata file.
This directive MAY appear any number of times when multiple metadata files are combined for processing a single spreadsheet.

define-context

This directive allows various context information to be provided about a document. This allows data and documents to be found and therefore reused and compared. This directive provides a "definition" rather than a "declaration" because it generally provides the data directly rather than pointing to where it exists inside the spreadsheet.

```
define-context <field> "<value>"|<reference>
```

`<field>`	A name that denotes the kind of context information being provided. This can be any UTF-8 String. This document provides a few suggestions for possible fields and the meaning of the values however, appropriate context information should be agreed within a community of practice and should be tailored to the specific user needs of that community. A registry of allocated field names is provided below. Users should take care to allocate field names that will not clash with those used by other communities of practice. Field names used outside of a single community of practice should be agreed with GDS and will be included in the registry below.
`<value>`	The value data for the specified field. This is always interpreted as a UTF-8 String. NOTE: The Tab Separated Value Serialisation of this language also requires its own escaping around BACKSLASH characters. Therefore, to embed a double quote in a `<value>`, it should be encoded as `\"` and to embed that in a Tab Separated Value Serialisation, it should be encoded as `\\"`.
`<reference>`	A single cell, specified using the standard Excel cell reference format (i.e `A1`) or a named range that can be expected to be declared by the spreadsheet data file (i.e `Sheet1!NamedRange`). A range MUST describe a single cell only. This is the cell in the spreadsheet that contains the value data for the

specified field. The contents of this field MUST be a UTF-8 String and will be extracted, validated and parsed with the same primitive as data cells declared as type String.

When this directive is encountered, the implementation should add the specified definition to a dictionary that is made available to the user. If the specified definition is already present in the dictionary then the implementation MUST fail with an error.

`field` Registry

Here we document the purpose of specific, named, context fields and the meaning of the values.

`doc-creator`
: The person that created the document.
This should be an eMail address in any of the RFC 822 mailbox or group formats however, the format of the first example below is preferred and recommended. For groups of maintainers, the last format is preferred and recommended.
For example:
- Gareth Heyes <gareth.heyes@digital.cabinet-office.gov.uk>
- andyjpb@digital.cabinet-office.gov.uk
- Andy (Data Team) <andyjpb@digital.cabinet-office.gov.uk>
- "Gareth (Data Team)" <gareth.heyes@digital.cabinet-office.gov.uk>
- Source Route <@home,@work:me@example.net>
- Data Peep: Andy Bennett <andyjpb@digital>
- Data Peeps: Gareth Heyes <gareth.heyes@digital>, andyjpb@digital;

`created`
: A date in RFC 3339 format. UTC is preferred, but not required. As RFC 3339 cannot represent symbolic timezones (other than 'Z'), the date specified MUST exist in the past at the time this piece of context information is defined.

`id`
: A document identifier in a locally defined format.
For example, it could be a four-part, hyphen separated identifier as follows:
- Unique ID for the area
- Department
- Sub Department if required
- Document type: CSV, ODS etc

`description`
: A short, human readable description or summary of the document that can be shown in search results.

require-context

This directive allows a spreadsheet receiver to specify that specific context information must be provided for a document.

`require-context <field>`

`<field>`	This is defined the same as for the `define-context` directive. NOTE: The Tab Separated Value Serialisation of this language also requires its own escaping around BACKSLASH characters.

When a certain field is named by a `require-context` directive, a corresponding `declare-context` directive MUST appear. The `require-context` directives usually appear in the receiver's part of the metadata and the `define-context` directives usually appear in the provider's part of the metadata but this is not a requirement. If, by the end of metadata processing, there are context fields that have been specified by a `require-context` directive and not supplied by a `define-context` directive then the implementation MUST fail with an error.

define-format-hint

This gives the parser a hint about the encoding of the spreadsheet file.
`define-format-hint <format>`

`<format>`	This MUST be one of following values `xls`, `xlsx`, `ods`, `rfc-4180`, `csv`, `tsv`.

The supplied spreadsheet SHOULD be in the specified format but the implementation MUST NOT fail to process the spreadsheet if it would have succeeded had the directive not been given. The implementation MAY emit a warning if the spreadsheet is not in the specified format. This MAY be implemented as a no-op.
This directive is not a security primitive. Other measures MUST be taken when the implementation is expected to be exposed to untrusted user input either in the metadata files described by this document or by the spreadsheets themselves.
This directive MAY appear once but MUST NOT appear more than once in each metadata file.
This directive MAY appear once but MUST NOT appear more than once when multiple metadata files are combined for processing a single spreadsheet.

empty-rows

This directive tells the implementation what to do when it finds empty rows whilst extracting data from the region of cells specified by `declare-data`.
`empty-rows preserve|collapse|warn|error`

`preserve`	Include empty rows in the returned data.
`collapse`	Ignore empty rows.
`warn`	Emit a warning when empty rows are found and then ignore them.
`error`	Emit an error when empty rows are found and fail.

This directive MAY appear once but MUST NOT appear more than once in each metadata file.
This directive MAY appear once but MUST NOT appear more than once when multiple metadata files are combined for processing a single spreadsheet.

Pragmas

Pragmas control the configuration of the language parser itself. This allows flexible and fine-grained feature support without having to adopt a brittle version numbering scheme. By default the parser should support the basic version of the language in its original form. The appearance of pragmas is used to enable new features on a case-by-case basis as and when they are required. Pragmas can also be used to modify the behaviour of existing features.
So that we do not end up in a situation where we are unable to upgrade from the basic, original version of the language, we define a single pragma here that can be used to enable new, but as yet undefined, features.

pragma

This is a directive that tells the parser to perform the action identified. If the action is not understood then the parser does not support the required functionality. When this occurs, the implementation MUST stop processing the metadata file and fail with an error. This directive MUST occur in the stream before all other directives and MUST NOT occur after any other directive has been processed by the parser.

```
pragma <action>
```

`<action>`	The action to perform or feature to enable. This standard does not specify any actions but any validly encoded argument can appear as this standard specifies enough to decode it.

Extracting Data

- Ensure the implementation can handle all of the declared data types.
- Resolve any named range references.
- Analyse the declare-header and declare-data directives to work out whether the data is organised into rows or columns. We assume that the header appears above or to the right of the data. There can be empty space between the header and the data but if the header is a row then the columns of the header MUST correspond to the columns of the data and if the header is a column then the rows of the header MUST correspond to the rows of the data.
- Extract the data region and use the header types to validate, parse and extract the contents of each cell into an internal (either native or user-defined) data type in the implementation language.

Serialisations

Tab Separated Value Format

The file is

- a UTF-8 encoded text file
- without a Byte Order Mark
- usually with the file extension `.slang`
- containing a single directive, along with its arguments, per line
- where the name of each directive appears at the start of the line, followed by its arguments
- and the separator between the directive and arguments, and between each argument is a single tab character
- blank lines are ignored
- Unicode codepoints are encoded as UTF-8, except TAB, BACKSLASH and the C0 and C1 control characters which are encoded as the UTF-8 encoding of their C-style escape sequences. A BACKSLASH that is followed by a character that does not denote a C-style escape for TAB, BACKSLASH or a C0 or C1 control character MUST cause the parser to stop, return an error and fail.

Example: poc.slang

```
#       Example metadata file

declare-type    "Item"  String
declare-type    "Price" GBPxVAT
declare-type    "Quantity"      Number
declare-type    "Total" Formula
declare-header  A3:D3
declare-data    A4:D8
```

Example: Embedded Escapes

```
#     This declares a column header that contains some double
#     quotes.
declare-type    "My name is \\"Slim Shady\\""   String

#     This declares a column header that contains a BACKSLASH.
declare-type    "C:\\Documents" Windows-Pathname

#     This declaration contains an invalid escape sequence and
#     should fail at the Tab Separated Value Format unwrapping
#     stage.
declare-type    "There's more to life than \"escaping\""   String
```

Embedded Workbook Format

The file is an Excel or OpenDocument Workbook and

- the first sheet is dedicated to the Spreadsheet Description Language Metadata
- the directives appear one per row, starting from row 1
- empty rows are ignored
- the directive appears in column A

- the arguments appear one per column from column B rightwards
- range references must specify the name of the sheet to which they refer, regardless of whether they are named references or regular references

Example: poc.slang

#	Example metadata file	
declare-type	"Item"	String
declare-type	"Price"	GBPxVAT
declare-type	"Quantity"	Number
declare-type	"Total"	Formula
declare-header	Sheet2!A3:D3	
declare-data	Sheet2!A4:D8	

Example: Embedded Escapes

The workbook container takes care of delimiting the cells for us so we only need to manually perform the escaping that is defined in the directive definitions.

#	This declares a column header that contains some double	
#	quotes	
declare-type	"My name is \"Slim Shady\""	String
#	This declares a column header that contains a BACKSLASH.	
declare-type	"C:\Documents"	Windows-Pathname

Security Considerations

Implementations may be exposed to untrusted user input either in the metadata files described by this document or by the spreadsheets themselves. Implementations should take steps to validate untrusted user input and ensure that it is safe before processing it. The language defined by this document is limited in scope and does not define or require a way for untrusted code to be executed directly.