Fielded Text Standard

1 Document

1.1 Information

a) Author

Paul Klink

b) Copyright

c) Release Date

22 Mar 2015

d) Status

Draft

e) Document Version

0.8.1

f) Fielded Text Standard Version

1.1

1.2 Change Information

a) Version 0.8.1

- Description of Culture has been significantly improved.
- Minor improvements to other descriptions in document.

b) Version 0.8

- Added EndOfLineIsSeparator
- Added DateTime format strings (Preliminary)
- Document Formatting and description improvements.
- Minor changes to terms.

2 Introduction

2.1 Definition

Fielded Text is a standard which provides structure and schema definition to text files which contain tables of values. Typically, a line in these text files represents a record (or row) and the line itself is divided into fields (or columns). Comma Separated Variable (CSV) files and Tab Delimited files are examples of Fielded Text Files.

2.2 Purpose

The purposes of the Fielded Text standard are:

a) To make it easier to parse files containing fielded information (ie tables of values).

b) To assist in generating files which conform to a specified format.

c) To increase the chances of the information in a file being correctly interpreted.

d) Improve methods of transferring and exchanging tables of data.

e) Allow files containing fielded information to be accessed with database like techniques.

2.3 Compatibility

The Fielded Text standard aims to be compatible with nearly all existing files containing fielded information. It supports the following capabilities to provide compatibility with existing files:

a) Fields separated by a delimiter character,

b) Fixed Length Fields,

c) Mixed Fixed Length and Delimiter separated fields in a line,

d) Quoted Fields (Optional or Explicit),

e) New Lines in Quotes,

f) Automatic New Line detection or specified New Line character,

g) End Of Line as record separator (instead of terminator),

h) Comments,

i) Ignoring Blank Lines,

j) Ignoring extra characters/fields in lines,

k) Handling language cultures,

l) Character Encodings,

m) Multiple Heading Lines (both delimited and fixed length),

n) Substitutions (escape sequences),

o) Embedded (Stuffed) Quote Characters,

p) Boolean, DateTime, Decimal, Float, Integer and String fields,

q) Field Heading Constraints,

r) Null fields,

s) Constant fields,

t) Specify format and styles of fields,

u) Lines can have different sequences of fields based on the value of “key” fields.

3 Files

3.1 Files and Streams

In order to make this standard more readable, it refers to Fielded Text “files” and Fielded Text Meta “files”. However, in most cases, “file” can be respectively construed as any content or stream of data containing fielded information or XML data which is Meta for fielded text. For example, it can also represent: web pages, memory streams, other IO streams etc.

3.2 Fielded Text Meta Files

The “Meta” of a text file specifies the format of a fielded text file. It specifies items such as:

a) Fields in the file,

b) Sequence of fields in the file,

c) The format and type of each field,

d) What character is used to delimit fields,

e) The substitutions used in the file.

The Meta is specified in a XML file. The schema for this XML file is specified in this document.

3.3 Declared and Undeclared Fielded Text files

Fielded Text files can either be declared or not declared. Declared Fielded Text Files begin with 2 lines which contain the following information:

a) A special string which identifies the file as being a Fielded Text file.

b) The version of the Fielded Text Standard which this file conforms to.

c) A link to the Meta which specifies the format of this Fielded Text File

d) Custom/File specific information

Undeclared Fielded Text files do not begin with these two lines and do not contain the information. When these files are parsed, then Meta file needs to be explicitly specified. Existing CSV or Tab Delimited or other text files containing fielded information are undeclared Fielded Text files.

3.4 Character Encoding

The character encoding of Fielded Text files is not specified in the Meta for the files. For Fielded Text files, character encoding is considered a transport issue and outside the scope of this standard. The standard contains references to ASCII and Unicode characters however these are provided for example only.

Fielded Text Meta files are XML files and use XML conventions for specifying their character encoding.

3.5 Culture

Different countries and regions throughout the world have different conventions

concerning how to display dates and numbers and how to compare text. To support these different conventions, formatting and string comparison are always done within the context of a culture. This culture is specified by the "Culture" property in the Meta. The value of this property is specified in accordance with RFC 4646 however it is not case sensitive.

String comparisons (case sensitive and case insensitive) are performed in accordance with the language conventions of the specified culture. The culture also specifies various parameters used in field formatting. These parameters are described in Culture Parameters (5.3). The language rules and parameters always reflect the conventions of the country or region. Generally these are not expected to change, but if the conventions do change in a country/region, then the language rules and parameters for the corresponding culture will also change accordingly.

There is one culture which is not dependent on any country or region and is guaranteed not to change. This is the "Invariant Culture". The parameters associated with this culture are specified in "Invariant Culture Parameter Values" (5.3.1). If cultural considerations are not important, then it is recommended that the invariant culture be used. It provides the highest reliability that the data will be correctly interpreted as its parameters are well defined and are not subject to change.

If the Culture is not specified in the Meta or its value only contains a zero length string, then the Invariant Culture is to be used.

3.6 Whitespace

Whitespace Characters are printable characters which provide horizontal and vertical separation and do not provide a visual mark. The selected culture determines whether a character is a whitespace character however the following characters are always considered as whitespace:

a) Horizontal Tab (ASCII 9)

b) Line Feed (ASCII 10)

c) Vertical Tab (ASCII 11)

d) Form Feed (ASCII 12)

e) Carriage Return (ASCII 13)

f) Space (ASCII 32)

g) Whitespaced categorised Unicode Characters.

Whitespace characters can have special significance in certain contexts within a Fielded Text file. For example, a Line Feed character may mark the end of a record. The special uses of whitespace characters will be further covered throughout the rest of this document.

3.7 Lines

A Fielded Text file consists of 0 or more lines. How a line starts and ends, and the types of lines in a fielded text file is described below.

3.7.1 Start and End of a Line

A line starts with either the first character in a file or the first character after the End of Line (EoL) character(s) that flagged the end of the previous line. A line ends with the first EoL character(s) encountered after the start of the line. The last line in the file can either end when the EoL characters are encountered or when the end of the file (EoF) is reached. The EoL character(s) are not considered part of the line.

3.7.1.1 EndOfLineIsSeparator

The EoL character(s) can also be used to separate lines rather than terminate them. The "EndOfLineIsSeparator" Meta property specifies if EoL character(s) are line separators. If this property is True, then the last line in the file must not be followed by the EoL character(s).

3.7.1.2 EndOfLineType and EndOfLineChar

The "EndOfLineType" Meta property determines which character(s) are the End of Line character(s). This property can have one of three values:

a) "Auto"

Auto specifies that any of the following 3 character sequences will flag the end of a line:

- <Carriage Return> (ASCII 13)
- <Line Feed> (ASCII 10)
- <Carriage Return><Line Feed> (ASCII 13 followed by ASCII 10)

b) "Char"

Char specifies that the EoL character is specified by the "EndOfLineChar" Meta property. The "EndOfLineChar" property must contain a string with one character only in it. This character becomes the EoL character.

c) "CrLf"

CrLf specifies that the character sequence <Carriage Return><Line Feed> (ASCII 13 followed by ASCII 10) flags the end of a line.

3.7.1.3 EndOfLineAutoWriteType

When generating files, the "EndOfLineAutoWriteType" Meta property specifies which character(s) should be used to mark the end of a line if the "EndOfLineType" is "Auto". The "EndOfLineAutoWriteType" property can have one of the following values:

a) "CrLf"

The EoL characters are <Carriage Return><Line Feed> (ASCII 13 followed by ASCII 10).

b) "Cr"

The EoL character is a <Carriage Return> (ASCII 13).

c) "Lf"

The EoL character is a <Line Feed> (ASCII 10).

d) “Local”

One of the above EoL character sequences is used based on the local environment / operating system. This value ensures that any Fielded Text files generated on a computer can be correctly viewed with other programs on that same computer.

3.7.2 Line Types

A Fielded Text file contains 4 types of lines:

a) Heading Line

A Heading line is a line in the Heading Section of a Fielded Text file. These lines correspond to Headings often placed at the top of files containing fielded information.

b) Record Lines

These lines only exist in the Body Part. They contain the actual fielded information. Typically, one line represents one record (or row) of fielded information. However if a field contains EoL character(s) (and the “AllowEndOfLineCharInQuotes” Meta Property is True), then it is possible for a record to span multiple lines.

c) Comment Lines

Fielded Text files can contain comment lines. Any line which is not a Heading Line or a Record Line or an ignored Blank Line (see below), needs to be a comment line. This allows Fielded Text files to be easily parsed by other types of Parsers.

A Comment line is a line which begins with the “Comment Character”. (There are possible 2 exceptions to this, Heading Lines and New Lines embedded in a quoted field; these are covered later in this document). The “Comment Character” is either specified by the Meta or is specified by the Declaration part (if it is present).

Comment lines are generally ignored in a Fielded Text file however in 2 sections (Declaration and Embedded Meta) they are not ignored.

d) Ignored Blank Lines

Blank lines are treated like any other non-comment line if the “IgnoreBlankLines” Meta Property is False. However if this Meta Property is True, then Blank Lines become Ignored Blank Lines and are ignored. (Similarly to Comment Lines, there are possible 2 exceptions to this, Heading Lines and New Lines embedded in a quoted field; these are covered later in this document).

4 Structure

4.1 Overall

At the highest structural level, a fielded text file consists of 2 parts: Header and Body. Both parts are optional. In fact an empty file can be a valid Fielded Text

file. The Header part always comes first and is immediately followed by the Body part.

4.2 Header Part

The Header itself consists of up to five sections. These are:

a) Declaration

b) Comments

c) Embedded Meta

d) Comments

e) Heading Lines

Each of these sections are optional. If they are present, it will be in the above order.

The Declaration section, if it is present, will be the first 2 lines in the file. The first character in the first line of the declaration becomes the “Comment character”. This overrides the Comment Character specified in the Meta. Each of these 2 lines in the declaration part begin with this comment character.

If the declaration specifies that the Meta should be loaded from either an external file or an external URL, then this Meta will be downloaded immediately after the 2 declaration lines. The new Meta takes affect for the 3rd line onwards. If the new Meta specifies a different Comment character, then this also will take affect from the 3rd line onwards.

The Comments and Embedded Meta sections can contain comment lines or ignored blank lines. Both Comments sections are ignored when parsing.

The Embedded Meta section will be present if the Declaration specifies that the Meta for this Fielded Text file is embedded within the file. This part contains the XML which is the Meta for this Fielded Text file. The lines containing a standard XML stream (conforming to the Fielded Text Meta Schema) however each line begins with the comment character and optionally followed by some white space characters. The start and end of this section is determined by the detection of specific XML tags/elements.

If an Embedded Meta is present, then this new Meta will take affect in the line that immediately follows the Embedded Meta section. If the new Meta specifies a different Comment character, then this also will take affect after the Embedded Meta section.

The number of Heading Lines contained in the header is specified by the “HeadingLineCount” property. If there is one or more Heading Line, then the first Heading Line will be the first line that is not a comment line or an ignored blank line.

The remaining Heading Lines will immediately follow the first heading line. If any of these remaining Heading Lines are blank, then they are still considered to be heading lines – even if “IgnoreBlankLines” Meta property is True. Also, if any of the remaining Heading Lines begin with a comment character they are also still considered to be heading lines (and not comments). In other words, the Heading Lines are always consecutive.

If the Meta specifies that there are no Heading Lines then the Body part starts where the Heading Lines part would have started.

4.3 Declaration Section

4.3.1 Signature and Comment Character

A Declaration Section is contained within a Fielded Text file if the following 2 conditions are met:

a) The file contains at least 17 characters, and

b) The 2nd to 17th characters match the Fielded Text Signature string.

The signature string is "|!Fielded Text^|" (not including the surrounding quote characters and with a space character between "Field" and "Text").

If the above conditions are met, then the "Comment Character" is set to the first character in the file.

4.3.2 Declaration Parameters

The rest of the first line and all of the second line consist of Declaration Parameters.

A Declaration Parameter is a name/value pair. The name/value pairs have a format of: <name>=<value>. The value part of the name/value part is a string which is always encoded with standard string quoting (see 5.1.5). The name part can be any string provided it does not contain the "=" character and it cannot start or end with a white space character. The name part is not case sensitive. The name and value parts are separated by an "="character. Any white space characters between the "=" character and the name or the value, are ignored.

Declaration Parameters themselves are separated by one or more white space characters. The first declaration parameter can either immediately follow the Signature string or be separated from the Signature string by one or more white space characters.

4.3.3 Version Declaration Parameter

The first declaration parameter must always be the Version declaration parameter. This parameter must always be present in a Declaration. (No other parameters are compulsory.) The Version parameter specifies the version of the Fielded Text standard which a Fielded Text file conforms to.

The name of this parameter is: "Version".

The value of the Version parameter has a format of:
<Major Version Number>.<Minor Version Number>[.[Any string not containing white space characters]]
The "Major Version Number" and "Minor Version Number" must always be present. The "Minor Version Number" can optionally be followed by a period and then (optionally again), a string. The string cannot contain white space characters.

A Fielded Text parser should use the version parameter to determine if it supports the version of the Fielded Text standard a file conforms to.

4.3.4 Meta Reference Declaration Parameters

A Meta Reference Declaration Parameter can be included to associate a Fielded Text file to its Meta. There are 3 different types of Meta Reference Declaration Parameters:

1) MetaEmbedded parameter (name: “MetaEmbedded)

   If this parameter exists, then the Meta for the Fielded Text file is embedded within the Fielded Text file. The value is ignored however it is recommended that the value be set to True.

2) MetaFile parameter (name: “MetaFile”)

   The value for the MetaFile is a path of a file which contains the Meta for the Fielded Text file.

3) MetaUrl parameter (name: “MetaUrl”)

   The value for the MetaUrl parameter is a URL where the Meta for the Fielded Text file can be downloaded.

Only one of the above 3 parameters should be included in the declaration. If more than one are included, then the one with the highest priority according to above list is to be used (1 = highest). If more than one of the same type (and highest priority) are included then the first of these in the file should be used.

4.3.5 Custom Declaration Parameters

Custom or user specific declaration parameters can be added to the declaration. The name of custom declaration parameters should not conflict with any existing declaration parameters defined by this standard. To avoid conflict with declaration parameters defined in future versions of this standard, custom parameters should have names beginning with “x-” or “X-” (not including quotation characters).

4.4 Embedded Meta Section

The Embedded Meta section only exists in a file if the declaration section contains the “MetaEmbedded” parameter. If this parameter is not present then even if lines representing the Meta are present, then they should be ignored (treated as comments). If the parameter is present, then the Embedded Meta section must be present otherwise the file is invalid.

All lines in the Embedded Meta section are comment lines. However if you remove the comment character that starts each line, then the lines make up the XML that represents the Meta for the file. After the comment characters have been removed from the lines, the resulting XML can be processed by any standard XML reader to get access to the Meta properties and elements.

The Embedded Meta Section is considered to be present if the string “<FieldedText” (not including quotes and case sensitive) is detected in a comment line in the header (but not in the declaration).

If the Embedded Meta section is determined to be present, then the start of the Embedded Meta section is determined as follows:

a) If a line prior to the line containing “<FieldedText”, contains the string “<?

xml ” (not including quotes; one space character at the end; and case sensitive) then the Embedded Meta section starts from this prior line. The prior line cannot be in the declaration section.

b) If no prior line contains the string “<?xml ”, then the Embedded Meta section starts from the line containing the “<FieldedText” string.

The end of the Embedded Meta section is detected by checking if a line contains the string “</FieldedText>” (not including quotes and case sensitive). The line that contains this string is the last line of the Embedded Meta section.

The XML which makes up the Meta starts from either:

a) The string “<?xml ” if it exists on the first line and is before the “<FieldedText” string.

b) Otherwise the XML starts from the “<FieldedText” string on the firstline.

All characters on the first line before the start of the XML are ignored.

All characters after the “</FieldedText>” string on the last line are also ignored.

All lines in the Embedded Meta section must be comment lines. If any line in this section does not begin with a comment character, then the file is invalid.

4.5 Comments Sections

Both comments sections only contain comment lines. These lines are always ignored when parsing.

4.6 Heading Lines Section

Heading Lines contain fielded data. However the values in these lines are the headings for the fields. Each field has an array of heading values. These values can describe the contents and purpose of a field or they can provide other supplementary information about the field.

The number of headings in a field is the same as the number of Heading lines. Both counts are therefore specified by the “HeadingLineCount” Meta property. The values in each successive heading line corresponds to equivalent indexed values in the field's array of headings.

If one or more of the heading lines is specified, then it is also necessary to specify which of these lines contains the “Main” heading. The “MainHeadingLineIndex” Meta property specifies the index of the main heading line. For example, if the first line held the main heading, then “MainHeadingLineIndex” would be set to 0. The main heading line is used in conjunction with Heading Constraints (see Heading Constraint, 4.10.4).

4.7 Body Part

The Body Part starts immediately after the Heading Lines section, or if there are no Heading Lines, where the Heading Line section would have started.

This part contains 3 types of lines:

a) Record Lines

b) Comment Lines

c) Ignored Blank Lines

Comment Lines and Ignored Blank lines within the Body are ignored when parsing.

Record Lines contain the fielded information. Normally one line represents one record (or row). However it is possible for a record to span multiple lines. This will occur if the EoL character(s) are embedded in a field. In this case, EoL characters do not designate the end of a record and are treated like other characters. EoL characters are allowed to be embedded in a field if the following 2 conditions are met:

a) The “AllowEndOfLineCharInQuotes” Meta property is True, and

b) The field is quoted. That is, the field is surrounded by quote characters.

If an EoL characters are embedded in a field, then the subsequent line is a record line – even if it starts with a comment character or it is blank line. Comment Lines and Ignored Blank Lines cannot occur within a record that spans multiple lines.

4.8 Record

Records contain the fielded data in a fielded text file. Each record contains an ordered set of values which correspond to the fields and sequences defined in the Meta. Typically a record corresponds to one line in the fielded text however it can span multiple lines as described above.

The Meta defines the Fields which a record can contain. The order of the fields in a record is defined by the Sequences in the Meta. If no sequences are defined in the Meta, then an implicit sequence is used. The implicit sequence consists of the list of all fields in the order they are declared in the Meta.

Each value in a record must conform to the properties of its corresponding field in the record. If there is only one sequence or the implicit sequence is used, then the list of fields will be the same for all records. If there is more than one sequence in the Meta, then it is possible for records to have different lists of fields. This is further described in the Sequences (4.13) section.

Fields can either be “Fixed Length” or “Delimited”. If a field is “Fixed Length”, then the subsequent field begins immediately after the last character in this field. If a field is “Delimited”, then the Delimiter Character (specified by the “DelimiterChar” Meta property) determines when the field ends. The first Delimiter character after the start of the field, which is not embedded in the field, flags the end of the field. The last character of the field is the one prior to the delimiter character. The subsequent field starts immediately after the delimiter character.

The “IgnoreExtraChars” Meta property determines if any extra characters can occur in a line after the last field in a record. If this property is False, then the EoL character(s) must immediately occur after the last field in a record. However, if it is True, then all characters after the last field are ignored until the EoL character(s).

If the last field in a line is a delimited field, then if the “IgnoreExtraChars” Meta property is False, then the Delimiter Char must not end this field. Since it is the

last field in a line, the EoL characters will end this field. If the "IgnoreExtraChars" Meta property is True, then the last field can be ended by either the Delimiter Char or by the EoL character(s).

The last field in the last line of a file can also be ended by the End Of File (EoF). If the last line does not end with the EoL character(s), then the EoF places the same role as the EoL in ending the last field. If the "EndOfLineIsSeparator" Meta property is True, then the last line cannot end with the EoL character(s).

4.9 Order of fields in a Record

The order of fields in a record is determined by sequences (see Sequences, 4.13). However a Meta may not specify any sequences. In this case the order of fields in a record is determined by the "implicit sequence". The implicit sequence is itself determined from the resolved order of fields using:

a) The order of the Field elements in the Meta, and

b) The value of the "Index" property in a field Element.

See "Calculating resolved order of elements" (4.15) for details of how a resolved order is determined from a list of elements.

The resolved order of fields forms the implicit sequence.

4.10 Field

A field specifies the type of data a value in a record contains. It also specifies how that data is formatted. Each field is defined in the Meta with a "Field" element.

4.10.1 Field Name

Each field has a name. The "Name" field Meta property specifies a field's name. Names do not have to be unique within a Meta however it is recommended that they be unique. Field names are also not case sensitive. When searching for a field by name, the search should be case insensitive and use the culture specified by the Meta. Also note that a field's name can be a blank string.

The main heading in a field can also be a field's name. If the "NameIsMain" heading constraint is selected then:

a) When parsing a fielded text file, the field name is set to the value in the field's main heading.

b) When generating a fielded text file, a field's name is used as the field's main heading.

4.10.2 Field Id

Each field has an "Id" Meta property. This property holds a 32 bit signed integer. The purpose of an Id is to allow fields to be identified via an integer instead of a name. This makes it easier to use fields in switch/case statements. However the Id can be used for other purposes.

4.10.3 Headings

The Meta can contain the heading values expected for a field. Each field's expected headings are stored in “Headings” property in the Field's Meta as standard String CommaText. The actual headings themselves are in the Fielded Text file however these values can be validated against the heading values stored in the Meta. A field has as many headings as specified by the “HeadingLineCount” Meta property, however the Meta does not need to store that many headings. If less are stored in the Meta, then the remaining headings are set to empty strings and if more are stored, then the excess are ignored.

4.10.4 Heading Constraint

A field's “HeadingConstraint” property specifies how the a fields headings in the Fielded Text file should be validated when the Fielded Text file is parsed. This property can have one of 5 values:

1) “None”: No validation.
2) “AllConstant”: The field's headings in the Fielded Text file must match the heading values for that field stored in the Meta. The comparison is case sensitive.
3) “MainConstant”: The field's main heading in the Fielded Text file must match the corresponding heading value for that field stored in the Meta. The comparison is case sensitive.
4) “NameConstant”: The field's main heading in the Fielded Text file must match field's name. The comparison is case insensitive and uses the culture specified by the Meta.
5) “NameIsMain”: If this constraint is selected, then, when a Fielded Text file is parsed, the field's name is set to the field's main heading value. When a file is generated, the field's main heading is set to the field's name. This constraint allows field names to be dynamically assigned directly from the text file.

4.10.5 Field Text, Raw Value and Value

The data stored in a field for a record can be interpreted in several ways. This document refers to this data in the following ways:

4.10.5.1 Field Text

The Field Text is the complete string for a field as it exists in the file. For delimited fields, the string includes quote characters where fields are quoted and any whitespace around the quote characters. It does not include the delimiter character.

4.10.5.2 Raw Value

The Raw Value is the string for a field after it has been processed for quotes and substitutions. It is the string value which will be cast to the data type corresponding to the field's data type (eg. Integer, DateTime). When writing a text file, the Raw Value is the string that is obtained after converting the field's value to a string and then (when necessary) applying padding and truncation. For Fixed Width fields, the Raw Value is always the same as its Field Text.

4.10.5.3 Value

A field's data as a type corresponding to the field's “DataType” property.

4.10.6 Constant Fields

The “Constant” field Meta property specifies whether a field is constant. A Constant field has its value defined in the Meta. When a Fielded Text file is parsed, each value in the file for that field, must match the constant specified in the Meta. If it does not match, then a parsing error occurs. When a file is generated, the constant value specified in the Meta is always written out to the fielded Text file.

The constant value itself is stored in the “Value” field Meta property. This property is ignored if the “Constant” property is not True. The values themselves are stored in this property using their respective standard format. That is, the culture and the field's format and style properties do not affect how data is stored in the “Value” property. See “Standard Formatting” (5.1) for a description of standard formats for all of the field data types.

A Field can also specify the constant to be Null. If the field is constant, and the “Null” meta field property is True then the field value must always be Null. See “Field Data Types” (4.10.9) for the definition of when a field value is considered to be Null. Note that the “Null” meta property is ignored if the field is not constant. Also note that “Null” field meta priority takes priority over the “Value” field meta property.

4.10.7 Delimited Fields

If the “FixedWidth” field Meta property is False or not present, then the field is a Delimited field. Delimited fields can be of variable length and they are terminated with the first non-embedded Delimiter Character after the start of the field (or the EoL character(s) if the field is the last in the record).

A Delimited field can either be quoted or not quoted. If a field is quoted, then the first non-whitespace character in the field must be the Quote character which is specified by the Meta “QuoteChar” property. Also the last non-whitespace character prior to the terminating delimiter character, must be the Quote character. The characters between the enclosing quote characters will make up the Raw Value of a field (after being processed for embedded quotes and substitutions).

Any Delimiter characters between the enclosing quotes of a quoted field are not treated as delimiters. They are simply treated as characters in the raw field value.

The Quote character itself can be included within a quoted field if the “StuffedEmbeddedQuotes” Meta property is True. In this case, if there are 2 successive quote characters within a quoted field, then neither of these 2 characters is the ending quote character. Instead this pair of quotes represents one normal quote character within the raw value of the field. If the “StuffedEmbeddedQuotes” Meta property is False or not present, then the first quote of a successive pair would be treated as the end enclosing quote character for the field.

Quoted fields also allow EoL character(s) to be included in the raw value for a

field. If the “AllowEndOfLineCharInQuotes” Meta property is True, then any EoL character(s) which appear within a quoted field, do not flag the end of a record. Instead they are treated like normal characters and are part of the raw value of the field.

If a field is not quoted, then all the characters from the first to the one prior to the terminating delimiter character, are part of the raw value of the field (after being processed for substitutions). The “StuffedEmbeddedQuotes” and the “AllowEndOfLineCharInQuotes” Meta properties do not change the behaviour of any of the characters in a field that is not quoted.

The “ValueQuotedType” field Meta property determines whether a field in a record is quoted. This property can have one of three values:

1) “Never”

   Specifies that the field is never quoted.

2) “Always”

   Specifies that the field will be quoted unless it has a Null value. (Fields with a Null value are never quoted).

3) “Optional”

   When parsing a text file, if the first non-whitespace character is the Quote character, then the field is quoted. Otherwise it is not quoted.

When writing a text file a field will be quoted if the field is not Null and any of the following conditions are met:

a) The “ValueAlwaysWriteOptionalQuote” field Meta property is True.

b) The the first non-whitespace character in the Raw value (the text that represents the field's value) is the Quote Character.

c) The “ValueWritePrefixSpace” field Meta property is True and the first character in the Raw value is a whitespace character.

d) The Raw value contains the Delimiter character.

e) The Raw value contains any of the EoL character(s).

When generating a fielded text field, you can use the “ValueWritePrefixSpace” field Meta property to place an extra space in front of a delimited field. This visually increases the separation between the fields and may make the field easier to review. Prefixing a space can be done on a field by field basis and is activated by setting this field property to True. If a field has a Null value, then it will not be prefixed by a space – even if this property is True. Be careful when using this property with “String” data type fields which are flagged as Never quoted as the space will be considered as part of the field value when the generated file is parsed.

4.10.8 Fixed Width

If the field's “FixedWidth” Meta property is True, then the field always consists of the same number of characters. Unlike Delimited fields, Fixed Width fields are not terminated by a special character. The subsequent field immediately follows the last character of the field. Also, Fixed Length fields are never quoted and do not contain substitutions.

The “Width” field Meta property specifies the number of characters in the field. It must have a value that is one or greater.

If a field's value is Null (file value – not the field's constant value), then raw value written to the file will be a string of Null characters. The Null character is specified by the field's “ValueNullChar” Meta property. The length of this raw value will always be width of the field. Likewise, when reading the value of a FixedWidth field, if all the characters in the raw value are the Null character, then the value is Null.

When reading or writing the raw value for a Fixed Length field it is quite possible that the raw value does not exactly match the width of the field. Several field meta properties determine how to resolve the mismatch between the raw value length and the field width.

4.10.8.1 Padding Properties (raw value shorter than field width)

If the raw value is shorter than the field width, then the raw value will be padded so that its length becomes equal to the field's width.

The field's “ValuePadAlignment” Meta property indicates whether the padding will occur on the left or the right side of the raw value. This property can also be set to “Auto”, in which case the alignment will automatically be set to the default for each field data type.

The field's “ValuePadCharType” Meta property indicates how the padding character is to be selected. It can have one of the following 3 values:

1) “Auto”

   The padding character is automatically chosen based on the data type of the field. “Field Data Types” (4.10.9) identifies what character is used for each data type. Note that if “Auto” is selected, then the other Padding properties are ignored when reading files – that is, Padding properties are only used when generating files.

2) “Specified”

   The padding character is specified by the “ValuePadChar” field Meta property.on the data type of the field. Note that if “Specified” is selected, then the other Padding properties are ignored when reading files – that is, Padding properties are only used when generating files.

3) “EndOfValue”

   The first padding character after the (unpadded) raw value is the character specified by the “ValueEndOfValueChar” field Meta property. This character flags the end of the value. By selecting “EndOfValue”, the (unpadded) raw value will match the original (unpadded) raw value as the padding can be identified when reading and discarded. Unlike as in the “Auto” and “Specified” values, if “EndOfValue” is selected, then the other Padding properties are used for both reading and writing.

   If more than one padding character is required, then all padding characters after the first padding character are ignored when reading. When writing, it is recommended that the “Auto” character be used as the subsequent padding characters.

The above description of “EndOfValue” applies to Right aligned padding. If the padding is left aligned, then the first character before the value will be the character specified by the “ValueEndOfValueChar” field Meta property and all characters prior to this will be ignored when reading.

4.10.8.2 Truncate Properties (raw value longer than field width)

If the raw value cannot fit into the width of the field, then information will be lost when the file is generated. The field's “ValueTruncateType” Meta property specifies how the truncation is to be done. It can have one of the following values:

a) “Left”

The left side of the raw value is truncated.

b) “Right”

The right side of the raw value is truncated.

c) “TruncateChar”

The raw value is set to a string of repeated characters. The character is specified by the field's “ValueTruncateChar” Meta property. The length of the string matches the width of the field.

d) “NullChar”

The raw value is set to a string of repeated characters. The character is specified by the field's “ValueNullChar” Meta property. The length of the string matches the width of the field. If this value is selected, then the field value will be considered as Null when it is read.

e) “Exception”

If a raw value needs to be truncated, the software component or library writing the file should raise an exception. The higher level software can then use this exception to handle this condition.

Note that the Truncate properties are ignored when reading files – they are only applicable when writing files.

4.10.9 Field Data Types

A field can hold different types of data values. The type of data value a field has is specified by the field's “DataType” Meta property. Depending on the type of data a field holds, the field will also have some extra Meta properties which specifies the field's format and style. The format and style properties specify how conversion between the fields value and its raw value is carried out. This is described below in detail for each Field Data type.

Note that a field's value can also be Null – irrespective of its data type. A field's data value is always Null if it is a delimited field which is always quoted and its raw value only contains whitespace. For Fixed Length fields, the data value will be Null if all its characters are the Null Character. Some of the data types also have other conditions in which the field's value is Null. These are also described below.

A field's “DataType” Meta property can have the following values.

a) “String”

Specifies that the field's data is a string; that is, a series of characters. The string can be up to 2,147,483,647 characters in length.

The Auto Pad character is a space (ASCII 32) and the Auto Pad alignment is Right Pad.

b) “Boolean”

The field's data is either the boolean True or False. The field Meta properties “FalseText”, “TrueText” and “Styles” specify how to convert between the Field's Value and its Raw Value. This conversion is described in detail in “Field Formatting, Boolean” (5.2.1). If the Raw Value is an empty string or only contains only whitespace, then the Value is Null.

The Auto Pad character is a space (ASCII 32) and the Auto Pad alignment is Right Pad.

c) “Integer”

The field's data is a signed Integer. The range is that of a 64 bit signed integer. The field Meta properties “Format” and “Styles” specify how to convert between the Field's Value and its Raw Value. This conversion is described in detail in “Field Formatting, Integer” (5.2.2). If the Raw Value is an empty string or only contains only whitespace, then the Value is Null.

The Auto Pad character is a space (ASCII 32) and the Auto Pad alignment is Left Pad.

d) “Float”

The field's data is a Floating point number. The range and precision is that of a IEEE 754 Double Precision (64 bit) Floating Point number . The field Meta properties “Format” and “Styles” specify how to convert between the Field's Value and its Raw Value. This conversion is described in detail in “Field Formatting, Float” (5.2.3). If the Raw Value is an empty string or only contains only whitespace, then the Value is Null.

The Auto Pad character is a space (ASCII 32) and the Auto Pad alignment is Left Pad.

e) “Decimal”

The Decimal type is similar to the Float type however it is appropriate for financial calculations requiring large numbers of significant integral and fractional digits and no round-off errors. Decimal numbers support a range of 922337203685477.5808 to 922337203685477.5807 and can have up to 4 decimal places.

Similar to the “Float” data type, the field Meta properties “Format” and “Styles” specify how to convert between the Field's Value and its Raw Value. This conversion is described in detail in “Field Formatting, Float” (5.2.3). If the Raw Value is an empty string or only contains only whitespace, then the Value is Null.

The Auto Pad character is a space (ASCII 32) and the Auto Pad alignment is Left Pad.

f) “DateTime”

The field's data is a date and/or time. The date can range from 1 January 0001 to 31 December 9999. The time can be specified down to an accuracy of 10^{-7} seconds (100 nanosecond). The field Meta properties “Format” and “Styles” specify how to convert between the Field's Value and its Raw Value. This conversion is described in detail in “Field Formatting, DateTime” (5.2.6). If the Raw Value is an empty string or only contains only whitespace, then the Value is Null.

The Auto Pad character is a space (ASCII 32) and the Auto Pad alignment is Left Pad.

4.11 Heading Lines

Heading Lines are similar to records in that they consist of successive fields of values. However in this case, each value is a heading for the field. They are parsed and generated similarly to records however with the following differences.

a) Each heading field value is a string – regardless of the data type of the field.

b) A Field's value becomes one of the headings for that field. The index of the Heading Line determines which heading in the field is set to the value.

c) A heading line can be blank. If a heading line is blank, all its corresponding heading values in fields are empty strings.

d) Headings are formatted in the same way as string data type fields. The list below shows the field Meta properties which determine the formatting of heading fields. The equivalent property which affects the formatting of string fields in records is shown in brackets:

- “FixedWidth”
- “Width”
- “HeadingQuotedType” (“ValueQuotedType”)
- “HeadingAlwaysWriteOptionalQuote” (“ValueAlwaysWriteOptionalQuote”)
- “HeadingWritePrefixSpace” (“ValueWritePrefixSpace”)
- “HeadingPadAlignment” (“ValuePadAlignment”)
- “HeadingPadCharType” (“ValuePadCharType”)
- “HeadingPadChar” (“ValuePadChar”)
- “HeadingTruncateType” (“ValueTruncateType”)
- “HeadingTruncateChar” (“ValueTruncateChar”)
- “HeadingEndOfFieldChar” (“ValueEndOfFieldChar”)

Refer to the field sections for a description of how these properties affect field formatting. The above properties behave in exactly the same way for Heading fields as they for record fields.

e) The values of Heading Fields are subject to the field's “HeadingConstraint” property.

f) Heading lines can only have fields as specified by the root sequence or the implicit sequence. Sequence redirections cannot occur in heading lines.

4.12 Substitutions

4.12.1 Use of Substitutions

Substitutions allow strings and characters to be represented in a Fielded Text file with tokens. This can be useful in circumstances such as:

a) Field values contain non-printable characters.

b) Long strings are used repeatedly within a text file

c) Field values contain characters which have special purposes within a fielded text file.

d) A fielded text file is within a context where particular strings should not be present (for example, XML).

Substitutions are analogous to escape sequences as used in the C programming language though not exactly the same.

4.12.2 Identifying Substitutions

Substitutions exist within Field Text as pairs of characters. The first character of the pair is the Substitution Character. This character is analogous to the Escape Character in 'C' escape sequences. The Meta property “SubstitutionChar” specifies which character is the substitution character for the file.

The second character is the token character. If a substitution is defined in the Meta which specifies this token, then this substitution will be applied at this point in the file. If no substitution has this token, then the token character itself is the substitution.

Note that substitutions are only enabled if the “SubstitutionsEnabled” Meta property is True. If this property is not True, then the Meta “SubstitutionChar” property is ignore and all substitutions defined in the Meta are also ignored. Also, substitutions do not occur in Fixed Width fields.

4.12.3 Substitution Elements

The substitutions are specified by “Substitution” elements in the Meta. There is one element for each substitution. A Substitution element has the following Meta properties:

a) “Token”

Specifies the character which is the token for this substitution.

b) “Type”

There are several different types of substitutions. This property can have one of the following values:

- “String”

The substitution pair (Substitution character and token) are replaced by a string. The string is specified by the Substitution's “Value” Meta property

- “AutoEndOfLine”

  The substitution pair are replaced by the EoL character(s) specified by the “ EndOfLineAutoWriteType” main Meta property. Note that if the “ EndOfLineAutoWriteType” has a value of “Local”, then these substitutions will place EoL character(s) into the text that are easily readable on the local computer.

- “XmlCharLT”, “XmlCharGT”, “XmlCharAmp”, “XmlCharApos”, “XmlCharQuot”

  These 5 types will respectively replace a substitution pair with one of the following 5 characters: <, >, &, ', ". These characters are the 5 XML built-in character entities which cannot be included inside an XML element. If these substitution elements are included, then their respective characters will not be included in field text and it may be easier to include the text within XML elements.

c) “Value”

   Specifies the string or character to be substituted if the substitution type is “String”.

4.12.4 Processing Substitutions

Substitutions are processed when converting raw values to field text (writing) and when converting field text to raw values (reading).

When reading, if a string gets substituted into a field, then none of the characters in a substituted string (or the single character) have any special significance – they are simply treated as characters in the raw value. For example, if a character in a substituted string is the same as the character specified by the Meta “DelimiterChar” property, then that character is **not** treated as a delimiter. This also applies to characters which follow the substitution character but are not included as a token in any substitution elements.

When writing, the raw value is scanned for strings or characters which can be substituted. Where-ever possible, strings or characters in the raw value which match one of the substitution elements, should be replaced by that substitution. If at any point in the processing, more than one substitution can be applied, then the one declared earlier in the Meta will have priority and be used.

4.13 Sequences

Sequences define the order in which fields occur in records.

With Sequences, you can specify Fielded Text files where records can contain different fields based on the value of a key field. For example, an accounting Fielded Text file may specify both clients and transactions. If the value in the second field holds the value “C”, then the line (record) specifies a client; if the second field contains the value “T”, then the line specifies a transaction. All Client lines contain the same fields and all Transaction lines contain the same fields however fields in Client lines differ from the fields in Transaction lines.

To support these types of files, all different sequence of fields in a Fielded Text file need to be identified. In the above example, there would be 3 sequences defined:

1) First 2 fields of all records (Root),
2) Fields after the first 2 fields in Client records (Client),
3) Fields after the first 2 fields in Transaction records (Transaction).

One sequence is defined as the Root sequence. All records begin with the root sequence. In the above example, the sequence consisting of the first 2 fields is the Root sequence.

In addition to defining an ordered list of fields, a sequence can also associate one or more “Sequence Redirects” with a field. A Sequence Redirect contains a condition and the name of a sequence. If a field is associated with Redirects, then for each record in a file, the field's value is compared against the all the field's Redirect conditions. If the value matches a Redirect's condition, then the fields in the Sequence specified by the Redirect are invoked. That is for that record, the next and successive fields will be those fields listed in the sequence identified by the Redirect. Alternatively, the fields can be invoked after the current sequence comes to an end.

In the above example, the 2nd field of the root sequence will have 2 Redirects. One Redirect will have the condition of match “C” and specify the Client sequence. The other Redirect will have the condition of match “T” and specify the Transaction sequence.

Any sequence can invoke a new sequence with a redirect; not just the root sequence. Accordingly, a record can be subjected to multiple sequence redirects. It is also valid for one sequence to be invoked multiple times in the same record.

The Meta does not have to explicitly define any sequences. If no sequences are defined, it will be implicitly inferred from the order of the “Field” elements in the Meta. Typically, if all records in a Fielded Text file contain the same fields in the same order, then the sequence is not explicitly defined and the implicit sequence is used.

4.13.1 Sequence Elements

Sequences are specified by “Sequence” elements in the Meta.

Each element is identified by the Sequence's “Name” Meta property. All Sequences need to have a unique name. However sequence names are not case sensitive. So if 2 sequences have names that only differ by the casing of letters, then these names are not unique.

Only one sequence can have the sequence Meta property “Root” with a value of “True”. This defines that sequence as the Root sequence. Whenever a record (or heading line) is parsed, the root sequence is used to determine the order of the fields from the 1st field onwards. Note that for records (but not heading lines) other sequences can be invoked during the parsing of a records fields, if any of the field values match a redirect condition.

If none of the sequences have a “Root” property set to True, then the root

sequence will be the first declared in the Meta which does not have a “Root” Meta property.

Each sequence has an ordered list of sequence items. Each Sequence Item specifies a field and optionally specifies one or more Sequence Redirects associated with that field.

A Sequence's items (SequenceItems) can be defined in 2 ways:

1) Sequence Item elements

   The “SequenceItem” element is used to define the items. The implicit order of these items is the order they appear in the Meta. However a Sequence Item can explicitly specify its position in the order with its “Index” property.

2) FieldIndices property

   Sequence Elements can also be specified by the FieldIndices property. This property contains an array of Integers in Standard Integer CommaText format (5.1.7). Each integer is the index of the a field (in the resolved fields order).

   The array specifies a list of Sequence Items, each of which has the field specified by the Integer value. These Sequence Items cannot have an “Index” property so their position is always specified by the implicit order. They also do not have any redirects.

   The resolved order of the items is determined by the implicit order of those specified by elements, followed by those specified by the “FieldIndices” property. Items specified by an element, however can have their position explicitly set. This is further explained in “Calculating resolved order of elements” (4.15).

If the Meta does not contain any sequences, then the order of fields is determined from the “implicit sequence”. The implicit sequence is determined from the resolved order of fields as described in “Calculating resolved order of elements” (4.15).

4.13.2 Sequence Item elements

A Sequence can have zero or more Sequence Item elements.

Each Sequence Item must specify an existing Field. If a Sequence Item was specified with a “SequenceItem” element, then the element must contain the “FieldIndex” property. This property specifies the index of a field using the fields' resolved order.

If an Item was specified with its Sequence's “FieldIndices” property, then its (resolved) field index will be its Integer value in the “FieldIndices” integer array.

Sequence Item's specified by an element can also have an “Index” property. The value of this property explicitly specifies the Items position in its Sequence's resolved order of Sequence Items.

Sequence Items can contain zero or more Sequence Redirects. A Sequence Redirect is specified by a “Redirect” element within the Sequence Item element.

4.13.3 Sequence Redirect elements

A Sequence Redirect is used to change the order of fields in a record. A Redirect specifies a condition, a sequence and an invokation delay. If the field's value matches the condition, then the fields in the sequence specified in the redirect will be used in the record. The Redirect can specify that the new fields should be invoked immediately after the current field or they can be invoked after the current sequence is finished.

A record can have more than one redirect invoke a new sequence.

The Redirect Meta property “SequenceName” specifies the name of the sequence a redirect will invoke.

The Redirect Meta property “InvokationDelay” specifies whether the fields in the new sequence are to be invoked immediately after the current field or after the current sequence is finished. If a Redirect specifies that new fields should be invoked after the current sequence ends, then any Redirects in subsequent Sequence Items in the current sequence are to be ignored.

A sequence item can have more than one redirect. When checking for a match, the first redirect whose condition matches is applied. The redirects have a resolved order. This order is determined by the order in which they appear in the Meta (the implicit order) and whether a redirect has an “Index” Meta property. The “Index” Meta property explicitly specifies the order position for that sequence redirect. “Calculating resolved order of elements” (4.15) details how the resolved order is determined from implicit element order and the “Index” property.

The “Type” Redirect Meta property specifies what type of condition a Meta contains. It can have one of the following values:

a) “ExactString”

   The field value needs to match the string stored in the Redirect's “Value” Meta property. The comparison is case sensitive.

b) “CaseInsensitiveString”

   The field value needs to match the string stored in the Redirect's “Value” Meta property. The comparison is not case sensitive and uses the culture specified by the Meta “Culture” property.

c) “Boolean”

   The field value needs to match the Boolean value stored in the Redirect's “Value” Meta property. The Boolean is stored in the Meta using Standard Boolean formatting.

d) “ExactInteger”

   The field value needs to exactly match the Integer value stored in the Redirect's “Value” Meta property. The Integer is stored in the Meta using Standard Integer formatting.

e) “ExactFloat”

   The field value needs to exactly match the Float value stored in the Redirect's “Value” Meta property. The Float is stored in the Meta using

Standard Float formatting.

f) “ExactDateTime”

The field value needs to exactly match the DateTime value stored in the Redirect's “Value” Meta property. The DateTime is stored in the Meta using Standard DateTime formatting.

g) “Date”

The date portion of the field value needs to exactly match the date portion of the DateTime value stored in the Redirect's “Value” Meta property. The Time portions are ignored in the comparison. The DateTime is stored in the Meta using Standard DateTime formatting.

h) “ExactDecimal”

The field value needs to exactly match the Decimal value stored in the Redirect's “Value” Meta property. The Decimal is stored in the Meta using Standard Float formatting.

If a Sequence Item contains a Redirect condition whose data type differs from the Sequence Item's field's data type, then the comparison should be undertaken with the following steps:

1) Try to convert the field's value to the same data type as expected by the Redirect condition. If this conversion fails, then the field value is considered as not matching the condition.
2) Carry out the comparison with the converted field value.

It is recommended that Sequence Items use Redirects which expect the same data type as provided by the Sequence Item's field. Different implementations of Fielded Text may carry out Data Type conversions in different manners – leading to different interpretations of the data in a fielded text file.

4.14 Design Only Meta

4.15 Calculating resolved order of elements

The order of the elements in the Meta determines the implicit order, however “Index” property can override the position in the implicit order.

5 Value Formats and Styles

Various types of values need to be converted to and from text representations in both Fielded Text files and Fielded Text Meta files. These conversions are specified by formatting. There are 2 types of formatting:

1) Standard Formatting (5.1)

   A simplified formatting used in Fielded Text Meta Files and Declaration section in Fielded Text files.

2) Field Formatting (5.2)

   A more comprehensive formatting used by fields in Fielded Text files.

5.1 Standard Formatting

Standard Formatting allows you to express values as strings using a simplified fixed format which is not affected by styles or culture. The Invariant culture is always used when comparing text or determining characters of special significance.

In Standard Formatting only the following characters are considered as Whitespace characters:

a) Horizontal Tab (ASCII 9)

b) Line Feed (ASCII 10)

c) Vertical Tab (ASCII 11)

d) Form Feed (ASCII 12)

e) Carriage Return (ASCII 13)

f) Space (ASCII 32)

g) Any whitespace categorised Unicode character

The following Data Types and Data Type arrays have standard formats:

5.1.1 Boolean

Standard formatted Booleans can be one of the following 2 string values:

1) “True”

2) “False”

These values are not case sensitive and are compared with the invariant culture.

5.1.2 Integer

Standard formatted Integers have the following format:

[ws][sign]digits[ws]

Items in square brackets ([and]) are optional; and the values of the other items are as follows.

a) ws

A series of white space characters.

b) sign

A culture invariant positive sign or negative sign character.

c) digits

A sequence of digits ranging from 0 to 9.

5.1.3 Float

Both Float and Decimal numbers can be stored in the Standard Float Format.

Standard formatted Float numbers have the following format:

[ws][sign]integral-digits[.[fractional-digits]][e[sign]exponential-digits][ws]

Optional items are framed in square brackets ([and]). Items containing the term "digits" consist of a series of numeric characters ranging from 0 to 9. The values of the items are:

a) ws

   A series of white space characters.

b) sign

   A culture invariant negative sign or positive sign symbol.

c) integral-digits

   A series of digits specifying the integral part of the number. Integral-digits can be absent if there are fractional-digits.

d) '.'

   A culture invariant decimal point symbol (a period).

e) fractional-digits

   A series of digits specifying the fractional part of the number.

f) 'e'

   An uppercase or lowercase character 'e', indicating exponential (scientific) notation.

g) exponential-digits

   A series of digits specifying an exponent.

5.1.4 DateTime

Standard formatted DateTimes have one of the following formats:

a) yyyyMMdd

   Specifies a DateTime where the time portion is either 0 or irrelevant. It consists of an 8 character string containing a 4 digit year, a 2 digit month and a 2 digit day.

b) hhmmss

   Specifies a DateTime where the date portion is irrelevant. It consists of 6 character string containing a 2 digit hour, a 2 digit minute and a 2 digit seconds.

c) yyyyMMddhhmmss

   Specifies a DateTime with both date and time portions. The first 8 characters specify the date as described above. The next 6 characters specify the time as described above.

d) yyyyMMddhhmmss.<digit>[0 to 6 <digits>]

   Specifies a DateTime with fractional seconds. The first 8 characters specify the date as described above. The next 6 characters specify the time as described above. This is then followed by a period character (ASCII 46) and at least one digit (ranging from 0 to9). The digits after the period specify the fractional seconds. Up to 7 digits can be used to specify the fractional seconds – allowing time to be specified to an accuracy of 100

nanoseconds.

e) 'hhmmss.<digit>[0 to 6 <digits>]

Specifies a Time with fractional seconds. The date portion of the DateTime is irrelevant. The first 6 characters specify the time as described above. This is then followed by a period character (ASCII 46) and at least one digit (ranging from 0 to9). The digits after the period specify the fractional seconds. Up to 7 digits can be used to specify the fractional seconds – allowing time to be specified to an accuracy of 100 nanoseconds.

5.1.5 String Quoting

Standard String Quoting is implemented as follows:

a) The Quote character is always " (ASCII 34).

b) All existing Quote Characters in a string are replaced with 2 Quote Characters ("").

c) A Quote Character is placed at the start and the end of the string.

5.1.6 String CommaText

Standard String CommaText allows you to express an array of strings as a single string. It is implemented as follows:

a) A string in the array must be quoted with standard string quoting if it meets any of the following conditions.

- The string contains one or more Comma Characters (ASCII 44)
- The first non-white space character in the string is the Quote Character (ASCII 34).

b) Even if a string in the array does not meet any of the above conditions, it still can be optionally quoted with standard string quoting.

c) The resultant strings (after quoting if required) are then combined into one long string with separators. This is done by placing them in the string in their order according the array. The separator between each array string is one Comma character (ASCII 44).

d) If an array string is quoted, then optional white space characters can be placed between the enclosing quote characters and the Comma Character (or the start or end of the resultant string if the array string is respectively the first or last in the array). Otherwise, no other characters can appear in the string.

5.1.7 Integer CommaText

Standard Integer CommaText allows you to express an array of Integers as a single string. It is implemented as follows:

a) Each Integer in the array is converted to a string using Standard Integer formatting.

b) The resultant string array is then converted to a single string using Standard String CommaText formatting (5.1.6).

5.2 Field Formatting

Field formatting used used to convert between field values and the text representation. It uses styles and the culture specified by the Meta. Field formatting is far more comprehensive than Standard Formatting (5.1) and aims to support most text representations of values.

5.2.1 Boolean

5.2.1.1 Format

Two Meta properties determine whether a text string represents a True or False value:

1) The “FalseText” Meta property specifies the string which represents a value of False.
2) The “TrueText” Meta property specifies the string which represents a value of True.

When a text string is parsed, it is firstly compared against the Meta TrueText string. If these 2 strings match, then the value is True. If they do not match, then the text string is compared with the Meta FalseText string. If these match then the Value is False, otherwise there is a parsing error.

5.2.1.2 Styles

Styles provide some additional capabilities when parsing a text string. These styles only apply to parsing and not when generating strings.

a) IgnoreCase

   Case is ignored when text strings are compared to the Meta FalseText and TrueText strings. The selected culture is used when doing case insensitive comparisons.

b) MatchFirstCharOnly

   Only the first characters of the text string and the Meta FalseText and TrueText strings are compared.

c) IgnoreTrailingChars

   If the text string is longer than the Meta FalseText, then the comparison ignores all characters in the text string which are beyond the length of the Meta FalseText string. For example, if this style is selected then the text string “Failure” would match the Meta FalseText “Fail”. The same applies when comparing the text string with the Meta TrueText.

d) FalseIfNotMatchTrue

   If the text string matches the Meta TrueText, then the value is True. If it does not match, then the value is False. That is, the text string is not compared against the Meta FalseText.

The styles themselves are stored in the “Styles” Meta Property. This property contains the names of all applied styles as a Standard Comma Text string.

5.2.1.3 Writing

Either the string specified by the FalseText or TrueText Meta property is written to a file depending on the value of the boolean.

5.2.2 Integer

5.2.2.1 Format

The format of a string is as follows.

[ws][sign]digits[ws]

Items in square brackets ([and]) are optional; and the values of the other items are as follows.

a) ws

   An optional white space.

b) sign

   An optional sign.

c) digits

   A sequence of digits ranging from 0 to 9.

5.2.2.2 Styles

Styles either restrict or allow additional formatting on strings. The styles affect the parsing of fields but not the generating of fields.

Integers can have the Numeric styles applied as described in “Numeric Styles” (5.2.5). Note that not all numeric styles are applicable to Integers.

The styles themselves are stored in the “Styles” Meta Property. This property contains the names of all applied styles as a Standard Comma Text string.

5.2.3 Float

5.2.3.1 Format

When parsing floating point number strings, the basic format of a string is as follows.

[ws][sign]integral-digits[.[fractional-digits]][e[sign]exponential-digits][ws]

Optional items are framed in square brackets ([and]). Items containing the term "digits" consist of a series of numeric characters ranging from 0 to 9.

a) ws

   A series of white space characters.

b) Sign

   A negative sign or positive sign symbol.

c) integral-digits

   A series of digits specifying the integral part of the number. Runs of integral-digits can be partitioned by a group-separator symbol. (For example, in some cultures a comma (,) separates groups of thousands.) Integral-digits

can be absent if there are fractional-digits.

d) '.'

A culture-specific decimal point symbol.

e) fractional-digits

A series of digits specifying the fractional part of the number.

f) 'e'

An uppercase or lowercase character 'e', indicating exponential (scientific) notation.

g) exponential-digits

A series of digits specifying an exponent.

5.2.3.2 Styles

Styles either restrict or allow additional formatting on strings. The styles affect the parsing of fields but not the generating of fields.

Floating point numbers can have the Numeric styles applied as described in “Numeric Styles” (5.2.5). Note that not all numeric styles are applicable to floating point numbers.

The styles themselves are stored in the “Styles” Meta Property. This property contains the names of all applied styles as a Standard Comma Text string.

5.2.3.3 Writing

5.2.4 Decimal

5.2.4.1 Format

When parsing decimal number strings, the basic format of a string is as follows.

[ws][sign]integral-digits[.[fractional-digits]][e[sign]exponential-digits][ws]

Optional items are framed in square brackets ([and]). Items containing the term "digits" consist of a series of numeric characters ranging from 0 to 9.

a) ws

A series of white space characters.

b) Sign

A negative sign or positive sign symbol.

c) integral-digits

A series of digits specifying the integral part of the number. Runs of integral-digits can be partitioned by a group-separator symbol. (For example, in some cultures a comma (,) separates groups of thousands.) Integral-digits can be absent if there are fractional-digits.

d) '.'

A culture-specific decimal point symbol.

e) fractional-digits

A series of digits specifying the fractional part of the number.

f) 'e'

An uppercase or lowercase character 'e', indicating exponential (scientific) notation.

g) exponential-digits

A series of digits specifying an exponent.

5.2.4.2 Styles

Styles either restrict or allow additional formatting on strings. The styles affect the parsing of fields but not the generating of fields.

Floating point numbers can have the Numeric styles applied as described in “Numeric Styles” (5.2.5). Note that not all numeric styles are applicable to decimal numbers.

The styles themselves are stored in the “Styles” Meta Property. This property contains the names of all applied styles as a Standard Comma Text string.

5.2.4.3 Writing

5.2.5 Numeric Styles

Numeric Styles either restrict or allow additional formatting on strings. They are used when strings containing integers or floating point numbers are parsed.

There are 2 types of Numeric Styles: Basic and Composite.

1) A Basic Style specifies a particular restriction or enhancement to formatting.
2) A Composite Style specifies a set of Basic styles. Composite styles are provided to conveniently specify combinations of styles frequently used.

5.2.5.1 Basic Styles

a) AllowCurrencySymbol

The string is parsed as a currency if it contains the currency symbol. The selected culture determines which character is the currency symbol and how currencies should be parsed.

b) AllowDecimalPoint

The string can contain a decimal point. The selected culture determines which character is used as the decimal point.

c) AllowExponent

The string can be in exponential notation.

d) AllowHexSpecifier

The string contains a hexadecimal string. In addition to digits (0-9), strings can also contain characters A-F and a-f.

e) AllowLeadingSign

The number in the string can have a leading sign. The selected culture determines which character is used as the positive sign and the negative

sign.

f) AllowLeadingWhite

Any leading whitespace characters in the string are ignored when parsing.

g) AllowParentheses

The number in the string can be enclosed in one pair of parentheses.

h) AllowThousands

The number in the string can be separated into groups, for example a character can be placed between every 3 digits to show thousands. The selected culture determines how digits are grouped.

i) AllowTrailingSign

The number in the string can have a trailing sign. The selected culture determines which character is used as the positive sign and the negative sign.

j) AllowTrailingWhite

Any trailing whitespace characters in the string are ignored when parsing.

5.2.5.2 Composite Styles

a) Any

All styles are used except AllowHexSpecifier

b) Currency

All styles are used except AllowHexSpecifier and AllowExponent

c) Float

AllowLeadingWhite, AllowTrailingWhite, AllowDecimalPoint and AllowExponent styles are used.

d) HexNumber

AllowLeadingWhite, AllowTrailingWhite and AllowHexSpecifier styles are used.

e) Integer

AllowLeadingWhite, AllowTrailingWhite and AllowLeadingSign styles are used.

f) None

No styles are used.

g) Number

AllowLeadingWhite, AllowTrailingWhite, AllowLeadingSign, AllowTrailingSign, AllowDecimalPoint and AllowThousands styles are used.

5.2.6 DateTime

5.2.6.1 Format

The format of the DateTime in a string is specified by the “Format” Meta

property. The “Format” property contains a string which specifies how a DateTime is formatted as text. It is described in “DateTime Format string” (5.2.7) below.

5.2.6.2 Styles

Read Styles provide some additional capabilities when parsing a text string. These styles only apply to parsing and not when generating strings.

a) AllowInnerWhite

   Extra whitespace characters in the text string are ignored when parsing – unless the whitespace characters are specified in the format string.

5.2.7 DateTime Format string

DayOfMonth_NoLeading0, // d

DayOfMonth_Leading0, // dd

AbbreviatedNameOfDayOfWeek, // ddd

FullNameOfDayOfWeek, // dddd

10thOfSecond_Trailing0, // f

100thOfSecond_Trailing0, // ff

1000thOfSecond_Trailing0, // fff

10000thOfSecond_Trailing0, // ffff

100000thOfSecond_Trailing0, // fffff

1000000thOfSecond_Trailing0, // ffffff

10000000thOfSecond_Trailing0, // fffffff

10thOfSecond_NoTrailing0, // F

100thOfSecond_NoTrailing0, // FF

1000thOfSecond_NoTrailing0, // FFF

10000thOfSecond_NoTrailing0, // FFFF

100000thOfSecond_NoTrailing0, // FFFFF

1000000thOfSecond_NoTrailing0, // FFFFFF

10000000thOfSecond_NoTrailing0, // FFFFFFF

Era, // g gg

Hour_12_NoLeading0, // h

Hour_12_Leading0, // hh

Hour_24_NoLeading0, // H

Hour_24_Leading0, // HH

TimeZoneInformation, // K

Minute_NoLeading0, // m

Minute_Leading0, // mm

Month_NoLeading0, // M

Month_Leading0, // MM

AbbreviatedNameOfMonth, // MMM

FullNameOfMonth, // MMMM

Second_NoLeading0, // s

Second_Leading0, // ss

OneCharAmPmDesignator, // t

AmPmDesignator, // tt

2DigitYear_NoLeading0, // y

2DigitYear_Leading0, // yy

3DigitYear_Leading0, // yyy

4DigitYear_Leading0, // yyyy

5DigitYear_Leading0, // yyyyy

HoursOffsetFromUtc_NoLeading0, // z

HoursOffsetFromUtc_Leading0, // zz

HoursMinutesOffsetFromUtc, // zzz

TimeSeparator, // :

DateSeparator, // /

Literal, // / ' " other

5.3 Culture Parameters

The selected culture is used to set the following parameters used by Field Formatting (5.2)

a) Currency symbol character
b) Number decimal separator character
c) Negative sign character
d) Positive sign character
e) Number group separator character
f) Number group size character
g) Full day of week names
h) Abbreviated day of week names
i) Full month names
j) Abbreviated month names
k) AM/PM designator
l) One character AM/PM designator
m) Date separator character

n) Time separator character

Currently there is no well established standard for the determination of the value of these parameters for all the cultures identified by RFC 4646. The following document can be used as a guide https://msdn.microsoft.com/en-us/goglobal/bb896001.aspx to the parameter values for cultures.

5.3.1 Invariant Culture Parameter Values

The invariant culture is used when cultural conventions want to be avoided as far as possible. Standard formatting of number and date/time values is always performed within the context of this culture. The invariant culture is associated with the English language and uses the following formatting parameter values:

a) Currency symbol character: “$”

b) Number decimal separator character: “.”

c) Negative sign character: “-”

d) Positive sign character: “+”

e) Number group separator character: “,”

f) Number group size: 3

g) Full and abbreviated day of week names:

- “Monday”, “Mon”
- “Tuesday”, “Tue”
- “Wednesday”, “Wed”
- “Thursday”, “Thu”
- “Friday”, “Fri”
- “Saturday”, “Sat”
- “Sunday”, “Sun”

h) Full and abbreviated month names:

- “January”, “Jan”
- “February”, “Feb”
- “March”, “Mar”
- “April”, “Apr”
- “May”, “May”
- “June”, “Jun”
- “July”, “Jul”
- “August”, “Aug”
- “September”, “Sep”
- “October”, “Oct”
- “November”, “Nov”
- “December”, “Dec”

i) AM/PM and one character AM/PM designators:

- “AM”, “A”
- “PM”, “P”

j) Date separator character: “/”

k) Time separator character: “:”

It is recommended that the Invariant Culture be used when producing Fielded Text files and cultural issues do not need to be taken into consideration. These files are least likely to be affected by parsing errors due to incorrectly specified cultures or culture ambiguities.

6 Meta Reference

6.1 Root

6.2 FieldedText

6.3 Main/Root

6.4 General Attributes

6.5 Culture

6.6 CharEncoding

6.7 AutoDetectCharEncoding

6.8 EndOfLineType

6.9 EndOfLineChar

6.10 EndOfLineAutoWriteType

6.11 EndOfLineIsSeparator

6.12 QuoteChar

6.13 DelimiterChar

6.14 LineCommentChar

6.15 StuffedEmbeddedQuotes

6.16 SubstitutionEnabled

6.17 SubstitutionChar

6.18 AllowEndOfLineCharInQuotes

6.19 IgnoreBlankLines

6.20 IgnoreExtraChars

6.21 HeadingLineCount

6.22 MainHeadingLineIndex

6.23 HeadingConstraint

6.24 HeadingQuotedType

6.25 HeadingAlwaysWriteOptionalQuote

6.26 HeadingWritePrefixSpace

6.27 HeadingPadAlignment

6.28 HeadingPadCharType

6.29 HeadingPadChar

6.30 HeadingTruncateType

6.31 HeadingTruncateChar

6.32 HeadingEndOfValueChar

6.33 Design Only Attributes

6.34 NewBooleanFieldFalseText

6.35 NewBooleanFieldTrueText

6.36 NewIntegerFieldFormat

6.37 NewFloatFieldFormat

6.38 NewDateTimeFieldFormat

6.39 NewDecimalFieldFormat

6.40 NewBooleanFieldStyles

6.41 NewIntegerFieldStyles

6.42 NewFloatFieldStyles

6.43 NewDateTimeFieldStyles

6.44 NewDecimalFieldStyles

6.45 Elements

6.46 Field

6.47 Substitution

6.48 Sequence

6.49 Field

6.50 Attributes

6.51 DataType

6.52 Index

6.53 Id

6.54 Name

6.55 FixedWidth

6.56 Width

6.57 HeadingConstraint

6.58 Constant

6.59 ValueQuotedType

6.60 ValueAlwaysWriteOptionalQuote

6.61 ValueWritePrefixSpace

6.62 ValuePadAlignment

6.63 ValuePadCharType

6.64 ValuePadChar

6.65 ValueTruncateType

6.66 ValueTruncateChar

6.67 ValueEndOfValueChar

6.68 ValueNullChar

6.69 HeadingQuotedType

6.70 HeadingAlwaysWriteOptionalQuote

6.71 HeadingWritePrefixSpace

6.72 HeadingPadAlignment

6.73 HeadingPadCharType

6.74 HeadingPadChar

6.75 HeadingTruncateType

6.76 HeadingTruncateChar

6.77 HeadingEndOfValueChar

6.78 Headings

6.79 Null

6.80 Value

6.81 Format

6.82 Styles

6.83 FalseText

6.84 TrueText

6.85 Elements

6.86 Substitution

6.87 Attributes

6.88 Type

6.89 Token

6.90 Value

6.91 Elements

6.92 Sequence

6.93 Attributes

6.94 Name

6.95 Root

6.96 FieldIndices

6.97 Elements

6.98 Item

6.99 SequenceItem

6.100 Attributes

6.101 Index

6.102 FieldIndex

6.103 Elements

6.104 Redirect

6.105 Redirects

6.106 Attributes

6.107 Index

6.108 Type

6.109 SequenceName

6.110 InvokationDelay

6.111 Elements

7 Other standards and conventions

Following are some recommendations on how other standards and conventions can incorporate Fielded Text files

7.1 Mime Type

The Mime Type “text/fielded” be reserved for Fielded Text content

7.2 File Name Extensions

a) The extension “ftx” be used in names for files which contain Declared Fielded Text content.

b) The extension “ftm” be used in names for files which contain Fielded Text Meta data

8 Acknowledgements

.NET 1.1

9 Appendices

9.1 Appreviations

EoL	End Of Line