# Complex schema/format XML file

The "Complex Record" tools rely on a *complex schema* to interpret and process the record stream. In addition, the ComplexTextInput and ComplexTextOutput tools rely on a *complex format*, which is an extension of the complex schema, to know how to scan and format hierarchical text files.

This document is a reference for how to build and interpret complex schemas and complex formats.

## Detailed specification

There are two parts to the configuration of the input and output tools. These specifications are XML "dataitem" documents, and are stored in the repository or file system. The tools reference the schema and format specifications by path.

In this document, text is color-coded according to whether they apply to logical schema or formatting, as follows:

* Logical schema: BLACK
* General formatting: GREEN
* Complex-text (EDI, structured text, etc) formatting: ORANGE
* XML formatting: RED

There are multiple sections to the complex format/schema document:

* Global. This specifies the overall attributes of the schema and format, and supplies defaults when no finer-grained information is specified per record type. In many formats the treatment of records is completely uniform, and the global section is sufficient. This section describes overall formatting and scanning rules, as well as the "special modes" for X12 and EDIFACT.
* Per-record-type specification: This specifies the schema of each record type, and optionally specifies a record format to override the global record format.
* File record type. The file record is a pseudo-record that is used to convey information about that path from which data is read or to which is shall be written. This is especially useful when multiple files are read and written at once, using wildcards.
* Nesting structure. This specifies logical relationships between parent and child records and the nesting hierarchy that can be contained in the file. It starts with the "root" record type, which is implicitly the only child of the "file" record type.

The resulting schema/format document has this structure:

<m>

 <m k="global">

 <m k="datatypes">

 <!-- global datatype/validation/format definitions-->

 </m>

 <m k="format">

 <!-- global format -->

 </m>

 <m k="match">

 <!-- global match rule -->

 </m>

 <m k="garbage" type="string"/>

 </m>

 <l k="records">

 <m type="string" name="string">

 <l k="schema">

 <m name="string"

 required="boolean"

 group="string"

 typename="string"

 min\_length="integer"

 max\_length="integer"

 location="element|attribute"

 path="string"

 index="integer"

 ignore="boolean"

 null\_placeholder="boolean"

 offset="integer"

 width="integer"

 />

 ...

 </l>

 <l k="groups">

 <m required="boolean" name="string"/>

 </l>

 <m k="format">...record format...</m>

 </m>

 ...

 </l>

 <m k="root">

 <!--specifies nesting structure-->

 <!--is a level definition-->

 </m>

 <l k="roots">

 <!--alternate specification allows list of roots-->

 <m>

 <!--specifies nesting structure-->

 <!--is a level definition-->

 </m>

 ...

 </m>

</m>

## Records section

**records** is a map from record types to record definitions.

Within each **schema** element:

* **name** is the field name.
* **required** is equivalent to "not null".
* **group** is optional; if specified it is the name of the group that this field belongs to. The fields of a group should be contiguous. This option is only effective in delimited formats with a separator2 specified, as it allows all-null groups or trailing null fields of a group to be omitted.
* **typename** is the name of the type defined in the global datatypes section.
* **min\_length, max\_length** are as specified in the **validation** section of global datatypes. If specified they override any setting in the datatype named by **typename.**
* **offset, width, ignore, null\_placeholder, location, path, index** are as specified in the **format** section of global datatypes. If specified they override any setting in the datatype named by **typename**.

The **groups** contains properties of field groupings in the record. Each section contains:

* **name** is the name of the group
* **required** may be false to allow the entire group to be omitted. This allows individual fields within a group to be required only if the group is present. When scanning input, a group is considered "present" if one or more fields within the group is not null.

## Root and level definitions

The **root** item specifies which record type is the root (nested under the file record).

Under the **root** item are nested level definitions:

<m

 type="string"

 subtype\_field="string"

 subtype\_value="string"

 name="string"

 min\_count="integer" max\_count="integer"

>

 <l k="children">

 <!-- level definition -->

 ...

 </l>

</m>

Within each level definition:

* **type:** the typeof record. The type "FILE" is reserved and must not be used.
* **subtype\_field** and **subtype\_value** further differentiate the type of record based on matching a value found in a field. *They only exist to support X12 and other EDI formats and are ignored in other modes.*
* **name** optionally provides a name for this level that is unique with respect to all other children of its parent. If unspecified, **type** is assumed to be the name. In XML formats **name** defines the element tag for the record. This is used to disambiguate multiple uses of the same record type with distinct meanings (e.g. ShippingAddress and BillingAddress are both Address types). It is also used to uniquely name a level that can be referenced by other levels.
* the **min\_count** and **max\_count** number of occurances of the record at this position in the nesting structure. If **max\_count** is omitted, it is assumed to be infinite. If too few or to omany records of this type match within a given nesting level, warnings will be issued.
* **children** contains thelist of child-record relationships. May nest to an arbitrary depth.

## Record formats

Record formats are specified by the following section. This is used in the global record format and the per-record-type formats:

<m

 format="delimited|fixed|pattern|xml"

 comment="string"

 may\_skip\_level="boolean"

 end\_of\_level="boolean"

 omit\_trailing\_nulls="boolean"

 long\_record\_action="ignore|warn|error|fatal"

 decimal="string"

>

 <m k="match"

 kind="inherit|always|leading\_set|leading\_string|

 contains\_string|regex|parent\_field\_value"

 leading\_set="string"

 leading\_string="string"

 contains\_string="string"

 regex="string"

 limit="integer"

 negate="boolean"

 parent\_record\_type="string"

 parent\_field="string"

 parent\_field\_value="string"

 />

 <m k="end\_of\_record"

 kind="inherit|none|newline|string|regex"

 string="string"

 regex="string"

 skip\_trailing\_whitespace="boolean"

 />

 <m

 k="delimited"

 quote="string"

 escape="string"

 separator="string"

 separator2="string"

 trim="boolean"

 ignore="string"

 special\_mode="|edifact|x12"

 />

 <m

 k="fixed"

 width="integer"

 input\_short\_lines="boolean"

 />

 <m

 k="pattern"

 input="string"

 output="string"

 limit="integer"

 >

 <l k="groups" v="int|int|..."/>

 </m>

 <m k="xml"

 location="element|attribute"

 omit\_null\_attributes="boolean"

 omit\_null\_elements="boolean"

 omit\_null\_attribute\_placeholders="boolean"

 omit\_null\_element\_placeholders="boolean"

 prettiness="|tabs|space2|space4"

 />

 <l k="format\_list">

 <!-- list of padding, text, or fields sections in order -->

 <m section="text"

 value="string"

 optional="boolean"

 />

 <m section="padding"

 code="integer"

 length="integer"

 />

 <m section="fields">

 <l k="field\_names" v="f1|f2|..."/>

 </m>

 <m section="children"

 before="string"

 separator="string"

 after="string"

 stop="string"

 min\_count="integer" max\_count="integer"

 >

 <l k="types" v="c1|c2|..."/>

 </m>

 <m section="end\_of\_record"/>

 ...

 </l>

</m>

**format** specifies the formatting type, and is one of **xml, delimited, fixed**, or **pattern**.

If **comment** is encountered at the beginning of a record scan, the text up to and including the next end-of-record marker (see below) is discarded. Ignored if missing or blank.

**omit\_trailing\_nulls** only matters when **format** is **delimited**. If true, it indicates that trailing null fields of a record should be omitted entirely, including the separators. In a record with **groups**, trailing fields of a group that are null are omitted within the group. This mostly exists to support EDI standards.

**long\_record\_action** is processed when **format** is **delimited** or **fixed**. It indicates the action to be taken if the end-of-record marker is not found in the expected place. The default is **error**.

* **ignore**: skip forward to find end-of-record marker.
* **warn**: skip forward to find end-of-record marker, and issue a warning.
* **error**: skip forward to find end-of-record marker, and issue an error
* **fatal**: abort processing.

**may\_skip\_level** indicates that a nesting level may be omitted in the hierarchy. This is used, for example, in the UNG/UNE nesting level in EDIFACT. Omitted levels are turned into records with all fields set to NULL on input. Similarly, records with all nulls and may\_skip\_level set on output, will omit the record.

**end\_of\_level** indicates that this record is a "trailer" for its parent "header". Setting this to true indicates that this record type is the last child of its parent.

**decimal** is specified to change the numeric decimal point from its default of "." to something else (generally, "," for Europe-style).

### Match section

The **match** section is optional, and overrides the global match rule for this record type. Otherwise, the global match rule is used. Per-record match rules are useful when there is a deviation from the global match rule, especially when child records are embedded within parent records.

* **kind** defines the strategy to use for matching it is one of:
	+ **inherit**: Use the global match rule. This is the default.
	+ **always**: record is always scanned if context permits. This is normally used only in embedded-child records where the before/separator/after strings bracket the records, or where the child record count is strictly limited by the **min\_count** and **max\_count** settings of the **children** section.
	+ **leading\_set**: match one of a set of characters at the beginning of a field. The match is case-sensitive.
	+ **leading\_string**: match this string at the beginning of a field. The match is case-sensitive.
	+ **contains\_string**: look for this string somewhere within the field contents, up to limit
	+ **regex**: match the field to this leading regular expression, up to limit
	+ **parent\_field\_value**: look into the containing parent record of type **parent\_record\_type** in the nesting hierarchy, matching the value in **parent\_field** to the specified **parent\_field\_value**. The match is case-sensitive.
* **regex:** when **kind** is **regex**, thisis the regular expression to match (or not match). Regular expressions match anywhere in the text stream (up to the **limit**). If you want to match starting at the current position, start the regex with the '^' anchor character.
* **leading\_set:** when **kind** is **leading\_set**, this is a list of characters, any one of which should match at the beginning of the record.
* **leading\_string:** when **kind** is **leading\_string**, this is a string to match at the beginning of the record.
* **contains\_string:** when **kind** is **contains\_string**, this is a string to match anywhere in record text. Only usable when **end\_of\_record** type is **string** or **newline**.
* **negate** is true to invert the sense of the regular expression or character list match.
* **limit** indicates how far ahead to look when kind is **contains\_string** or **regex**. It defaults to 100. This number should be set to be the smallest value that will still match the record(s) in question.

### End-of-record section

The **end\_of\_record** section controls processing of the end-of-record marker.

* **kind** specifies the overall type of end-of-record marker. If unspecified, it defaults to **newline**. **kind** is one of the following**:**
* **inherit**: Use the global end-of-record setting. This is the default.
* **none**: Nothing is written on output at the end-of-record, and nothing is expected on input.
* **newline**: The OS-specific newline is written on output, and any newline is accepted on input.
* **string**: The **string** attribute is written on output at the end of every record, and expected on input.
* **regex**: The **regex** attribute is used to match and consume the end-of-record marker on input. The **string** attribute is used for the end-of-record on output.
* **skip\_trailing\_whitespace** is true to skip all whitespace characters following the end-of-record mark.

### Delimited section

The **delimited** section is used when **format** is **delimited**. The items not listed here are are interpreted as per CsvInput/Output. If omitted, the higher-level default setting is used.

* **quote** is a character that surrounds the data when it contains any special character such as a separator or quote character. If the quote char is empty, there is no quoting.
* **escape** may be used instead of **quote** to escape special characters (such as separators and the escape character itself) by preceding them with this character.
* **separator** is a string used to separate field values (currently must be a single character).
* **separator2** is used for delimited formats that contain substructure, such as EDIFACT and X12 (currently must be a single character). It specifies the second-level delimiter. Defaults to **separator** if unspecified.
* **ignore** is a string containing a set of characters to ignore completely on input.
* **trim** is true if leading/trailing whitespace should be trimmed from the CSV input data. This is optional and defaults to false.

**special\_mode** enables custom processing for known formats:

* **edifact** enables the following special behavior:
	+ Read/write UNA header and its terminator/separator/escape codes.
	+ UNB header and code page interpretation.
	+ BMG scanning and formatting
	+ format is "delimited"
	+ separator, separator2, and escape is set from UNA header on input
	+ decimal\_point is set from UNA header on input
	+ omit\_trailing\_nulls is true
	+ end\_of\_record/string set to segment terminator
	+ end\_of\_record/skip\_trailing\_whitespace is true
* **x12** enables the following special behavior:
	+ ISA header processed for separators and segment terminator
	+ GS header
	+ ST header
	+ decimal\_point, separator, and separator2 is set from ISA header on input
	+ omit\_trailing\_nulls option is true
	+ end\_of\_record/string set to segment terminator
	+ end\_of\_record/skip\_trailing\_whitespace is true

### Fixed section

The **fixed** section is used when **format** is **fixed**. If omitted, the higher-level default setting is used. It is interpreted as follows:

* **input\_short\_lines** allows an end-of-record sequence before all fields have been scanned as the end of those fields, and the remaining fields will get NULL values, without issuing a warning message.
* **width** specifies the width of the record excluding end-of-record marker. It is optional. It will be increased if necessary to fit all defined columns.

### Pattern section

The **pattern** section is used when **format** is **pattern**. This is *not* processed for the global format. It is interpreted as follows:

* **input** is a regular expression used to scan the input text. Subexpressions surrounded by () are successively mapped to the field name list. The number of () and the number of field names must match. A regex starting with ^ anchors the match at the current position. Without the ^, the regex can match anywhere up to the end-of-record marker. A regex ending in $ anchors the match at the end-of-record marker. Without the $, any text between the matched text and the end-of-record marker is skipped. See [http://www.regular-expressions.info](http://www.regular-expressions.info/) for a tutorial on regular expressions.
* **groups**, if specified, contains a list of one-based regular expression group numbers indicating which capture group is mapped to each field. If unspecified, the groups are mapped to fields in the order encountered. This is useful if your regular expression contains nested groups. Groups are numbered left-to-right as each opening "(" is encountered, so for example this regular expression:
 abc(def(ghi)\*jk)xx(yy)
contains three groups numbered (1,2,3). Given the input:
 abcdefghighijkxxyy
the groups will contain
 1: defghighijk
 2: ghighi
 3: yy
* **output** is a string with placeholders where formatted field values are inserted in the order in which fields are listed. Use the form \[FIELDNAME] where you want to output the value of FIELDNAME. For a literal \ use \\.
* **limit** indicates the longest input record that should be matched, including parts that are skipped. It defaults to 1000. Increase this limit to allow for very-long records.

### XML section

The **xml** section is used when **format** is **xml.** It contains information for formatting records as XML:

* location: Set to **element** to format field values as element text, otherwise field values will be formatted as attributes.
* omit\_null\_attributes: If true, null field values formatted as attributes will omit the attribute rather than setting the attribute to an empty string, unless a null placeholder value was specified in the datatype.
* omit\_null\_elements: If true, null field values formatted as attributes will omit the element rather than outputting empty element text, unless a null placeholder value was specified in the datatype.
* omit\_null\_attribute\_placeholders: If true, null field values formatted as attributes will omit the attribute rather than setting the attribute to the null placeholder, even if a null placeholder value was specified in the datatype.
* omit\_null\_element\_placeholders: If true, null field values formatted as attributes will omit the element rather than outputting the null placeholder, even if a null placeholder value was specified in the datatype.
* **prettiness** controls indentation of the XML document:
* if not specified, there is no indentation or newlines.
* **tabs** indicates newlines at end of elements, and indentation levels using tabs.
* **space2** indicates newlines at end of elements, and indentation levels using two spaces.
* **space4** indicates newlines at end of elements, and indentation levels using four spaces.

### Format list section

*Format list sections should not be specified in X12 and EDIFACT modes.*

**format\_list** is a list of one or more format specifiers:

* **text**: literal text to be copied to the output, or required on input.
* **padding**: padding characters to be output, or skipped on input.
* **fields**: A list of fields to scan/output. If the field name list is omitted, all fields are scanned/output.
* **children:** A list ofchild record types to scan/output here. If the list of child record types is omitted, all children are scanned/output here.
* **end\_of\_record**: the end of record marker will scan/output here.

#### Text format entry

* **value** is the text to scan/output
* **optional** is true to ignore non-matching text on input, in which case no text is consumed by this item.

#### Padding format entry

* **code** is the code point of the character to output (e.g. 32 for a space)
* **to\_column** is true to pad to the **column** position on the current line, false to pad a fixed number of characters.
* **length** is the number of characters to output when **to\_column** is false
* **column** is the column position when **to\_column** is true

#### Fields format entry

* **field\_names** is the list of fields to format, and the order in which to output them. If omitted, all fields are output in the order specified in the record schema.

#### Children format entry

A **children** format entry is optional. If not specified, all children will be output after the fields and end-of-record marker. Within this section:

* **types** contains the types of the child records to be formatted at this position. If **types** is omitted, all child record types are formatted here.
* **min\_count** and **max\_count** togetherspecify the total number of child that may be matched under this children section. These are optional; if unspecified, **min\_count** defaults to 0 and **max\_count** defaults to infinite.
* **before** is a string that occurs before the first child record and is omitted if there are no child records.
* **separator** is a string that separates adjacent child records.
* **after** is a string that occurs after the last child record. This is omitted if there are no child records.
* **stop** is a string that occurs after the last child record. This no effect on output, it is for input scanning only. The difference between **stop** and **after** is that the **after** string is consumed on input, but the **stop** string is not. Do not specify both **after** and **stop.**

#### End\_of\_record format entry

An **end\_of\_record** format entry is optional. If not specified, end-of-record will be output last, but before any implicit children.

## Global definitions

The global definitions section defines properties that apply to the entire schema or format.

<m k="global ">

 <l k="datatypes"> ... </m>

 <m k="garbage" type="string"/>

 <m k="format"> ... </m>

 <m k="match"> ... </m>

</m>

### Global datatypes

The global datatypes section defines named types and optionally, formats. You can define all of the type standards including basic formatting information in this section, and then refer to these predefined types by typename in the record definitions.

<m k="global">

 <l k="datatypes">

 <!--type definitions associated with names-->

 <m typename="string">

 <m k="datatype"...standard datatype.../>

 <m k="format"

 location="element|attribute"

 path="string"

 index="integer"

 ignore="boolean"

 null\_placeholder="boolean"

 offset="integer"

 width="integer"

 />

 <m k="validation" min\_length="integer" max\_length="integer" />

 />

 ...

 </l>

</m>

Within the **format** section:

* **offset** is the zero-based offset to the field when using fixed format.
* **width** is the width of the field when using fixed format.
* **ignore** is a list of characters to ignore when scanning values. For example when scanning currency this may be set to " £¥$,".
* **null\_placeholder** is a string, that when encountered on input, is replaced by null. On output, null will be replaced by this string. This helps deal with formats were e.g. 000000 is used instead of an empty string.
* **location**: specifies this field to be formatted as an XML element or attribute, if different than the record format defafult.
* **path**: specifies the element/attribute path, if different than the field name. This is case-sensitive, whereas the default mapping is not. Any '/' characters indicate further nesting of sub-elements, for example mapping the fields ShipAddr and ShipLast to paths Ship/Addr and Ship/Last corresponds to the following XML (assuming format/location="attribute"):
<Ship Addr="addr value" Last="last value"/>
* **index**: specifies a mapping from multiple fields to the same element name. For example, a record with Addr1, Addr2, Addr3 may format in XML as
<Addr>addr1 value</Addr>
<Addr>addr2 value</Addr>

<Addr>addr3 value</Addr>

The path for each field would be "Addr" but the format/index for Addr1 would be 0, Addr2 would be 1, etc.

Within the **validation** section:

* **min\_length** is the minimum length of a text value.
* **max\_length** is the maximum length of a text value.

### Global format

The global format specifies the default record format and some additional information about the encoding of the data:

<m k="global">

 <m k="format">

 ...

 </m>

</m>

**format** is a record format section, and specifies the global default record format.

### Global garbage

If a **garbage type** is specified, input not matching any record type may be treated as a "garbage" record, which may appear anywhere in the nesting hierarchy. The garbage record's fields and other properties must be defined like any other record.

<m k="global">

 <m k="garbage" type="string"/>

</m>

### Global match

The global match rule specifies a default rule for record types are recognized on input. This section is optional. If present, it defines the location of the record type value in the input records, from either a column range in a fixed-width file, or a field position in a delimited file. The record type value is always trimmed. This default match rule can be overridden on a per-record-type basis for records that do not follow a simple convention.

<m k="global">

 <m k="match">

 <!--specify one of delimited or fixed, to match global format-->

 <m k="fixed" column="integer" size="integer"/>

 <m k="delimited" position="integer"/>

 </m>

</m>

This is ignored when **special\_mode** is set.

**column** and **position** are zero-based.