Fragment Interchange

SGML Open Technical Resolution 9601:1996

Steve DeRose, EBT, Co-chair, Fragment Interchange Subcommittee, SGML Open
Paul Grosso, ArborText, Co-chair, Fragment Interchange Subcommittee, SGML Open

Revision date: 1996 November 7

Copyright © 1996 by SGML Open

Permission to reproduce parts or all of this information in any form is granted to SGML Open members provided that this information by itself is not sold for profit and that SGML Open is credited as the author of this information.

The SGML standard supports logical documents composed of a possibly complex organization of many entities. It is not uncommon to want to view or edit one or more of the entities or parts of entities while having no interest, need, or ability to view or edit the entire document. The problem, then, is how to provide to a recipient of such a "fragment" the appropriate information about the context of that fragment in the original document that is embodied in the part of the document that is not available to the recipient.

The goal of this resolution is to define a way to send fragments of an SGML document—regardless of whether the fragments are predetermined entities or not—without having to send everything up to the part in question. The delivered parts can either be viewed or edited immediately or accumulated for later use, assembly, or other processing. This resolution addresses the issues by defining:

1. exact constraints on what portions of an SGML document may constitute fragments to be supported by this resolution;
2. the set of information needed to allow for successful parsing as well as for viewing or editing of a fragment in a useful and important set of cases;
3. the notation (i.e., language) in which this information will be described;
4. some possible mechanisms for associating this information with a fragment.

Issues involved with the possible "return" of any such fragment to the original sender and the determination of the possible validity of the "returned" fragment in its original context are beyond the scope of this Resolution. While implementations of this Resolution may serve as part of a larger system that allows for "fragment reuse," the many important issues about reuse of SGML text are beyond the scope of this Resolution.

Technical Resolution 9601:1996
Committee draft: 1995 November 21
Committee draft: 1996 February 29
Final Draft Technical Resolution: 1996 July 31
Final Technical Resolution: 1996 November 7

1 Introduction

The need to make SGML documents available over the Internet is well known. This is easy as long as whole documents are sent, including their DTDs, SGML declarations, all entities, etc. But many SGML documents are too large to be managed by shipping them in their entirety when only a portion may be needed.

Many documents are megabytes in length, even excluding all the graphic, video, and other entities a document may reference. Transferring such a document can take too long for real-time access. Even after a document arrives, it may take too long to parse it and get to the desired part. If the user asked to look at chapter 20, one must parse 19 whole chapters before seeing it. With hypertext documents, one also can't afford to include every document the first one references, when the user will likely follow only a few of the links.

The obvious solution is to not send it all, but instead send things as they become needed. The goal of this resolution is to define a way senders can send small parts of an SGML document at need, without also having to send everything up to the part needed. This can be done regardless of whether the parts are entities or not, and the parts can either be viewed immediately or accumulated for later use, assembly, or other processing.

The SGML standard has some constructs that can be used to address these issues in certain situations. External text entities can be used, but they generally do not contain the necessary context information. Some tools and implementations, however, may be able to make use of such entities without the explicit context information. Furthermore, 8879 defines SUBDOC entities that are self-contained in terms of context (they are complete documents), but each SUBDOC forms its own ID name space and each must have its own DTD. Though some fragment applications can be addressed using the constructs already available in 8879, the constructs in the standard were not seen as being sufficient for all applications that need to use fragments. This Resolution was developed to provide an interoperable solution for fragment applications when the techniques of 8879 are insufficient.

The challenge is that an isolated element from an SGML document may not contain quite enough information to be parsed correctly. This resolution enables senders to provide the remaining information required so that systems can interchange any SGML elements they choose, from books or chapters all the way down to paragraphs, tables, footnotes, book titles, and so on, without having to manage each as a separate entity or having to risk incorrect parsing due to loss of context.

1.1 Scope

This resolution enables interchanging portions of SGML documents while retaining the ability to parse them correctly (that is, as they would be parsed in their originating document context), and, as far as practical, to be formatted, edited, and otherwise processed in useful ways. Specifically:

1. A sender can send a fragment that consists of any element or any sequence of SGML data that constitutes "mixed content" or "element content" drawn from an SGML document. Most commonly this means a sequence of contiguous sibling elements, but processing instructions, comments, whitespace, and certain other SGML constructs are also permitted.

Any element that begins within the fragment must end there as well, and any element that ends in the fragment must also start there (this constraint is sometimes called "being synchronous").

2. The fragment sent can be parsed correctly at the recipient end to produce precisely the same ESIS (SGML structure and content information) that the sender got when it parsed the fragment in its complete document context.
3. All capabilities of Basic SGML documents (except shortrefs) can be used in any fragment so sent, as well as variant capacities and quantities, as well as many variant delimiters and name characters.
4. Fragments can be sent exactly as they occurred in the original SGML data. Because they need not be changed in any way, it is possible to authenticate or validate that they have been received intact, and it is possible for users to cache them.

To accomplish these ends, this resolution defines:

1. exact constraints on what portions of an SGML document may constitute fragments to be supported by this resolution;
2. the set of information needed to allow for successful parsing as well as for viewing or editing of a fragment in a useful and important set of cases;
3. the notation (i.e., language) in which this information will be described;
4. some mechanisms for associating this information with a fragment.

Conceptually, a sender examines a fragment to be sent and, using the notation defined in this Resolution, constructs a fragment context specification. The object representing the fragment removed from its source document is called the fragment body. The sender sends the fragment context specification and the fragment body to the recipient. The storage object in which the fragment body is transmitted is call the fragment entity. (In some packaging schemes, the fragment context specification may also be embedded in the fragment entity.) The recipient processes the fragment context specification to determine the proper parser state for the beginning of the fragment and uses that information to put the SGML parser into the right state to be able to parse the fragment. The fragment body itself can then be parsed normally.

Issues involved with the possible "return" of any such fragment to the original sender and the determination of the possible validity of the "returned" fragment in its original context are beyond the scope of this Resolution. While implementations of this Resolution may serve as part of a larger system that allows for "fragment reuse," the many important issues about reuse of SGML text are beyond the scope of this Resolution.

1.2 Definition of a fragment

This Resolution defines a fragment to be the SGML representation of SGML data that constitutes either element content (SGML production [26]) or mixed content (SGML production [25]) extracted from a complete SGML–compliant document. The fragment shall be represented using at most the syntax and feature set of a Basic SGML document as defined in 8879, definition 4.22, except that:

1. the Core Concrete Syntax rather than the Reference Concrete Syntax shall be used (i.e., there can be no SHORTREFs), and
2. certain changes from the concrete syntax of Basic SGML documents to the capacities, quantities, delimiters, and name characters are permitted, and the FORMAL feature can be either YES or NO.

Variant delimiters and name characters may be used to the extent that they do not introduce conflicts with the delimiters required by this resolution. For example, accented or wide characters may be used freely, but the specific characters number sign (#), single (') and double (") quotation marks, parentheses (()), equal sign (=), and whitespace may not be added to the permitted SGML name characters because they could conflict with the use of those characters by this resolution.

2 Fragment context specification language

2.1 Formal syntax

A fragment context specification uses an extremely simple formal syntax which is chosen (a) to prevent delimiter conflicts if placing a fragment context specification inside an SGML file; (b) to ease the task of parsing fragment context specifications either with standard parser-generator tools or with handwritten programs; and (c) to reflect that a fragment context specification is information *about* SGML data, not SGML data itself. Though SGML syntax itself was considered as a possible syntax for the fragment context specification language, it was rejected on the basis of not being the best language for our purposes for a number of reasons, including complexities with delimiter conflicts, escaping issues, minimization, issues of being able to embed a string using SGML syntax within an SGML document, and so on.

Six delimiter characters are used in fragment context specifications, and they are shown as quoted literals in the grammar below. They have the same values regardless of what SGML declaration applies to the fragment itself (and its document context). Therefore variant concrete syntaxes in which those delimiter characters are added to the list of SGML name characters (LCNMSTRT, UCNMSTRT, LCNMCHAR, and UCNMCHAR) may not be used with this specification (variant concrete syntaxes that do not introduce such conflicts can be used freely).

Literals in the grammar shall be recognized without regard to case distinctions. Whitespace characters, represented in the grammar as "s", include space, tab, form feed, carriage return, and line feed.

Fragment context specifications use syntax that can be processed by a wide variety of commonly available parsing tools. That syntax is defined here combining the methods of lex and yacc, with these shorthand conventions (see John R. Levine, Tony Mason, and Doug Brown, *lex & yacc*, O'Reilly & Associates, Inc., 1990):

1. * and + are used throughout, not only at the lexical (lex) level. They indicate that the preceding token or sub-rule may be repeated; + indicates that at least one instance is required. Square brackets are also used as in lex (an initial "^" negates the list of permitted characters).
2. All characters other than the null character and the delimiters and whitespace already discussed are permitted as name characters in fragment context specifications.

The grammar described formally in the following section and generally in this document defines a fragment context specification language. Entities composed of this language can be said to be written in the SGML Open Fragment Context Specification Notation whose Formal Public Identifier is:

```
-//SGML Open//NOTATION Fragment Context Specification//EN
```

2.1.1 BNF Specification

```
fragspec : global* s* context
global   : "(" s* item s* ")"
context  : "(" s* "CONTEXT" s+ elemspec+ s* ")"
item     : "SGMLDECL" s+ dcl_loc
```

```
         | "DOCTYPE" s+ dtdcl_loc
         | "SUBSET" s+ external_id
         | "SOURCE" s+ external_id locator?
         | "LEVEL" (s+ attr)*
         | "COMMENT" (s+ value)*
         | "CURRENT" s+ gi (s+ attr)+
         | "LASTOPENED" s+ gi
         | "LASTCLOSED" s+ gi
         | "RESTATE" s+ revalue
         | extension (s+ attr)*
dcl_loc  : external_id
         | "WITHFRAGMENT"
         | "WITHSOURCE"
dtdcl_loc: name s+ external_id
         | "WITHFRAGMENT"
         | "WITHSOURCE"
external_id : "PUBLIC" s+ value (s+ value)?
         | "SYSTEM" s+ value
locator  : node (s+ dataloc)?
         | node s* "TO" s+ node
node     : s+ nameloc (s* treeloc)?
         | s+ treeloc
nameloc  : "(" s* "ID" s+ name s* ")"
treeloc  : "(" s* "TREELOC" (s+ number)+ s* ")"
dataloc  : "(" s* "DATALOC" s+ number (s+ number)? S* ")"
extension: "X-"namechar+
revalue  : "AFTERSTARTTAG" | "AFTERDATA" | "AFTERRSORRE"
         | "PENDINGAFTERRSORRE" | "PENDINGAFTERMARKUP"
elemspec : gi (s+ rep)? (s+ elemprop)* s* "(" s* elemspec* s* ")"
         | "#PCDATA"
         | "#FRAGMENT"
rep      : "#"number
elemprop : attr
         | "#NET"
         | "#MAP" s* "=" s* value

attr     : name s* "=" s* value
gi       : name
name     : namechar+
value    : "\'"[^']*"\'"
         | "\""[^"]*"\""
number   : [0-9]+
namechar : [^#()'"= \t\f\r\n]
s        : [ \t\f\r\n]
```

2.1.2 Examples

This example is intended to represent a typical case, which does not require many of the features needed to support particular SGML advanced features:

```
(DOCTYPE book PUBLIC "-//Acme//DTD Book//EN")
(SUBSET SYSTEM "c:\foo.ent")
(SOURCE SYSTEM "http://xyz.com/books/draft/b.sgm"
   (TREELOC 1 2 5 5 1))
(CONTEXT
 book version="draft" (
  fm()
  bdy (
    chp #4 ()
    chp label="5" (
      ct() sec #3 () sec ( #fragment ) sec #5 () )
    chp () )
  bm() )
)
```

The example below includes even cases that may be rare in practice:

```
(COMMENT "This fragment is subsection (4.4.1) of
  the book in galley form.")
(SOURCE SYSTEM "http://xyz.com/books/draft/b.sgm"
   (ID chap4) (TREELOC 1 5 1))
(DOCTYPE book PUBLIC "-//Acme//DTD Book//EN")
(SUBSET SYSTEM "c:\foo.ent")
(LASTCLOSED CT)
(LASTOPENED CT)
(CURRENT FIGR ent="myvalue")
(CURRENT P security="top")
(CONTEXT
 book version=draft (
  fm()
  bdy #net #map="map37" (
    chp #4 ()
    chp label="5" (
      ct() sec #3 () sec ( #fragment ) sec #5 () )
    chp () )
  bm() )
)
```

2.2 Item keywords

All items shall be used with the meanings explained in this section; the order in which they are specified is insignificant. It is an error to specify any item other than CURRENT, COMMENT, SOURCE, or an extension more than once. Should such an error be encountered, the last value specified shall be applied.

For correct processing, certain information must definitely be available to the recipient. Therefore a sender must either send those items, send references to them, or have reason to believe that the recipient already has them or knows how to find them. Such items include the SGML declaration, and all markup declarations needed for correct parsing. Few other items are needed

except when specific SGML capabilities are actually used: CURRENT items are only needed if #CURRENT attributes occur, attributes and sibling information are only needed for particular recipient processing such as auto-numbering or other formatting, and so on.

2.2.1 SGMLDECL: Reference to applicable SGML declaration

The SGMLDECL item may be included to indicate the SGML declaration applicable to the fragment's document or to specify that it can be found within the SOURCE document or fragment. There are several ways of indicating the declaration's location. The recipient shall determine what SGML declaration to use according to the following ordered list:

1. If SGMLDECL specifies the token WITHSOURCE, the SGML declaration should be included at the beginning of the storage object indicated by the SOURCE item's external id.
2. If SGMLDECL specifies the token WITHFRAGMENT, the SGML declaration should be included at the top of the fragment entity itself (except that it must follow the fragment context specification if one is embedded at the top of the fragment entity).
3. If the SGMLDECL item is omitted, the fragment-aware processor shall start to process the doctype declaration (as specified implicitly or explicitly via the DOCTYPE item). If an SGML declaration is found at the top of it, it shall be used.
4. If no SGML declaration is found via any of the above methods, then the receiving system shall apply any catalog resolution which it supports (e.g., the SGMLDECL and DTDDECL entries of an SGML Open TR9401 catalog).
5. If none of the above steps results in an SGML declaration, the receiving system shall apply its default implied SGML declaration.

2.2.2 DOCTYPE: Reference to applicable DTD

The DOCTYPE item specifies the DOCTYPE name for the document from which the fragment comes (such as "book") and the external identifier for the external subset of its DTD. This is typically obtained directly from the DOCTYPE declaration of the document. For example:

```
(DOCTYPE book SYSTEM "http://z.org/public/dtds/book.dtd")
```

Note: "Formal system identifiers" (or FSIs) as described in the "SGML General Facilities" annex of the present corrigendum to ISO/IEC 10744:1992 are one appropriate means of expressing system identifiers in this context; they can accommodate identifiers such as URLs.

The token WITHSOURCE as the value of the DOCTYPE item means that the storage object indicated by the SOURCE item's external id shall be inspected for an initial doctype declaration (optionally preceded by an SGML declaration if the SGMLDECL item is omitted) in exactly the form it would have been specified if the fragment were a complete document; if one is found there, this doctype declaration shall be used to process this fragment.

Similarly, WITHFRAGMENT means that the fragment entity (immediately following any fragment context specification that may be embedded at the top of the fragment entity) shall be inspected for an initial doctype declaration (optionally preceded by an SGML declaration if the SGMLDECL item is omitted), and if one is found there it shall be used to process this

fragment. In the case of both WITHSOURCE and WITHFRAGMENT, the doctype declaration may include an internal declaration subset.

If there is no DOCTYPE item, then (a) if there is a SOURCE item in this fragment context specification, the equivalent of `(DOCTYPE WITHSOURCE)` is assumed; (b) if there is no SOURCE item in this fragment context specification, the equivalent of `(DOCTYPE WITHFRAGMENT)` is assumed.

If the DOCTYPE is still not found, the results are implementation defined.

Note: In the case of WITHFRAGMENT, the presence of a DOCTYPE declaration in the fragment entity could allow a non-fragment-aware SGML parser to mistakenly attempt to parse the fragment entity as a complete document. If a system wishes to protect against any such possibility, it shall not include the DOCTYPE declaration at the top of the fragment entity.

2.2.3 SUBSET: Reference to applicable internal document type declaration subset

The SUBSET item specifies an external identifier for the internal document type declaration subset for the document from which the fragment comes or a sender-created portion of it (the [ ] delimiters are not to be included). This is typically obtained directly from the document type declaration subset of the document (if the information needed from the subset is not already in a separate SGML entity, the sender may create such an entity and assign it an external identifier).

SUBSET need not specify the entire document type declaration subset, but must specify enough of it to parse the fragment as it would have been parsed in the original, complete context. For example, it is permissible to omit general ENTITY declarations for entities that are not referenced or mentioned within the fragment, but not permissible to omit ones that are.

If the DOCTYPE declaration is provided at the top of the fragment entity (see WITHFRAGMENT above), then the subset must be provided there as well, and it is an error for a SUBSET item to appear in the fragment context specification; the correct error recovery is to ignore the SUBSET item.

2.2.4 LEVEL: What optional specification information is included

The LEVEL item enables senders to specify what optional information they are in fact including in the fragment context specification. Although optional information cannot change the way the fragment is parsed, it can be useful for other types of processing, such as formatting. The LEVEL item can contain several `name=value` pairs, from the set defined here. If any such pair is not present, the sender is deemed to not be specifying whether the corresponding information is included or not. Specifying names or values not in this list is an error, and the erroneous value shall be ignored. Specifying the same name more than once in the same LEVEL item is also an error, and the correct recovery is to accept the last occurrence.

1. FSIB: NO | SOME | LEFT | RIGHT | ALL

   This keyword may be used to state whether the CONTEXT item includes no siblings of the fragment, some siblings but not all, all left siblings, all right siblings, or all siblings.

2. ASIB: NO | SOME | LEFT | RIGHT | ALL

   This keyword works like FSIB, but identifies what siblings are provided for ancestors of the fragment, rather than for the fragment itself.

3. SATTR: NO | SOME | LEFT | RIGHT | ALL

   This keyword may be used to state what attributes are provided for siblings of the fragment: none, some but not all, all on left siblings, all on right siblings, or all on all siblings.

4. AATTR: NO | SOME | LEFT | RIGHT | ALL

   This keyword works like SATTR, but identifies what attributes are provided for ancestors of the fragment.

5. CONTENT: NODE | SIBLINGS | ELEMENT | MIXED

   This keyword may be used to state whether the fragment consists of a single element, a sequence of contiguous sibling elements, SGML element content, or more general SGML mixed content.

2.2.5 SOURCE: The identity of the fragment

The SOURCE item may be used to specify the origin or identity of the fragment sufficient for the recipient to request it again later, or to save a reference to it, or to do other contextual processing such as resolving IDREFs that point to elements outside the fragment. SOURCE is recommended in all fragment context specifications unless the application context makes it inapplicable (such as when no persistent identifier for the data exists or the document source is not accessible).

The **external_id** shall identify the entire document out of which the fragment was taken. The **external_id** can be any valid public or system identifier as defined by 8879. The locator shall identify the fragment element(s) within that document, using methods drawn directly from HyTime (ISO/IEC 10744:1992) and DSSSL (ISO/IEC 10179:1996). If the fragment consists of a single element (including its descendants), the TO clause of the locator shall not appear; if the fragment consists of more than a single element, then the TO clause shall appear: the locator before "TO" shall identify the first element or other node in the fragment, and the locator after "TO" shall identify the last element or other node in the fragment.

Note: Child nodes shall be counted as in the default DSSSL grove plan. "Child nodes" here means the items in the node list specified by the "content" property found on nodes of class "element". The node types used for content in the default DSSSL grove plan are: datachar, sdata, element, extdata, subdoc, and pi. Thus, the only nodes that count as children are those representing elements; processing instructions; SDATA, SUBDOC, and external data entity references; and characters in #PCDATA. Things such as comments, marked section boundaries, ignored REs, and ignored markup of any kind do not count.

In each locator, at least one of **nameloc** or **treeloc** shall appear:

1. The **nameloc**, if present, shall contain the value of the nearest ID attribute available either on the fragment's initial element or on an ancestor of it. If neither the fragment's initial element nor any ancestor has an SGML ID attribute, the **nameloc** parameter shall not be specified.

2. The **treeloc**, if present, shall contain a sequence of sibling numbers for walking down the document tree to the fragment, equivalent to the content of a **marklist** in a HyTime **treeloc** location address element. If **nameloc** is also specified, the element it locates shall be treated

as the location source where the walk begins; otherwise the document's root element is the location source. For example, to locate the second child of the fourth child of the root of the document specified by the **external_id**, the **treeloc** would contain "`1 4 2`".

3. The **dataloc** shall only be used when the fragment does not consist of SGML "element content" (essentially, when it does not consist of one or more complete elements, but includes #PCDATA chunks at its root level).

   Except that negative offsets may not be used, the offsets are equivalent to the content of a **dimspec** in a HyTime **dataloc** location address element whose quantum is "str" and whose location source is the element(s) specified by the adjacent **nameloc** and/or **treeloc** items. At least one of those items must be present whenever **dataloc** is present. The length parameter for the **dataloc** is optional because the receiving system can count the length for itself. The starting and ending offsets of a non-element-content fragment must point to locations directly within precisely the same SGML element.

2.2.6 COMMENT: User comments

A fragment context specification may include arbitrary comments using this item. The COMMENT item shall not be used for extensions intended to be processed by computer, for which the extension mechanism shall be used instead.

2.2.7 CURRENT: values for #CURRENT attributes

If the fragment uses no #CURRENT attributes, the CURRENT item is not needed. A current item must be included for every #CURRENT attribute whose value is not specified on its first occurrence within the SGML fragment (this is required even if a value for the attribute is also specified on some prior element mentioned in the fragment context specification, such as an ancestor). For example, given an attribute list declaration such as:

```
<!ATTLIST p
   type        NAME      #CURRENT
   secure      (y|n)     #CURRENT>
```

a fragment consisting of section 2 such as:

```
<chap>
  <sec>...<p type=4 secure=Y>Some text...</p></sec>
  <sec n=2><p>Some more text...</p></sec>
</chap>
```

contains a P element that must receive attribute values from a prior element outside the fragment. Therefore the fragment context specification for section 2 would include:

```
(Current P TYPE="4" SECURE="Y")
```

If multiple #CURRENT attributes are defined in the same SGML ATTLIST they may be either combined (as just shown) or listed separately (as shown below), with no change of meaning:

```
(Current P TYPE="4")
(Current P SECURE="Y")
```

Note: It is never necessary to indicate that a #CURRENT attribute has not yet been set before the fragment, because under SGML rules if that is true then the first occurrence within the fragment must have an explicit value.

The attribute value may generally be given either as the original value exactly as in the original SGML source, or may be the result obtained after parsing the value, case-folding it, and/or normalizing white space within it according to SGML rules. However, if the value contains an entity reference(s), then the value must be the exact source value, to ensure correct interpretation of entity reference(s) within the value.

If a #CURRENT attribute applies to a name group rather than to a single GI (as with the SGML ATTLIST declaration shown below), then each current item given for that attribute shall specify one of the GIs, not the entire name group. This is enough because the recipient has access to the DTD and can find the applicable ATTLIST and its name group.

```
<!ATTLIST (p | bq | fn)
   secure      (y | n)    #CURRENT>
```

A CURRENT item may be included for #CURRENT attributes that do not in fact occur within the fragment, and this is not an error. Senders should check and minimize what to transmit, but are permitted to send all the possibly-needed values without checking. It is an error to specify CURRENT more than once for the same attribute; should such an error be encountered, the last value specified shall be used.

2.2.8 LASTOPENED and LASTCLOSED: for empty start tags

If the fragment uses SGML empty start tags (<>) in certain ways, the fragment context specification must include the LASTOPENED and/or LASTCLOSED items:

1. LASTOPENED must be used to provide the GI of the last element opened prior to the fragment if OMITTAG is YES and the first element in the fragment begins with an empty start tag.
2. LASTCLOSED must be used to provide the GI of the last element closed prior to the fragment if (a) OMITTAG is NO, (b) an empty start tag occurs within the fragment, and (c) such a start tag occurs before any element happens to be closed within the fragment.

It is not an error to specify the LASTOPENED and/or LASTCLOSED items even if they are not actually needed. It is never necessary to send both. Implementors may choose to always send both, always send one (choosing which one based solely on OMITTAG), or check the conditions above and send these items only when actually needed.

2.2.9 RESTATE: record end handling state

An SGML parser implementing clause 7.6.1 of ISO 8879 has five distinct record-boundary processing states. The RESTATE item specifies which of these states is current at the start of the fragment. The following identifies these states by specifying one situation in which the parser enters this state; for each state, there are also other situations in which the parser can enter the state:

1. AFTERSTARTTAG: immediately after the start of a proper sub-element
2. AFTERDATA: immediately after data
3. AFTERRSORRE: immediately after an RS encountered in state AFTERDATA
4. PENDINGAFTERRSORRE: immediately after an RE encountered in state AFTERDATA

5. PENDINGAFTERMARKUP: immediately after a processing instruction encountered in state PENDINGAFTERRSORRE

If RESTATE is not sent, then modifying the fragment before the beginning of the first (or only) element of the fragment, after the end of the last (or only) element of the fragment, or between two elements at the top level of the fragment may not in all cases have unambiguous results. In some applications record boundaries in content may never occur or may have no significance, as determined by some application-specific semantic rules outside SGML. In such cases the RESTATE item may always be omitted.

2.2.10 extension: User enhancements

To add machine-processable information to fragment context specifications, a new item keyword may be created. Such a keyword must be named beginning with `X-`. A tool conforming to this Resolution must handle all such extensions (by processing those it recognizes and safely ignoring—while optionally emitting a warning message—those it does not recognize).

2.3 CONTEXT and its keywords

The CONTEXT item is required in all fragment context specifications and provides information about the element context of the fragment such as the list of element types open when it begins. It is the last item in any fragment context specification. The keywords described in this section appear when applicable within individual element specifications, rather than as freestanding items. In order to avoid potential conflict with attribute names, they all begin with "#" (which is the RNI delimiter in the Reference Concrete Syntax).

Parentheses in the CONTEXT item express tree structure from the SGML document from which the fragment came. Ancestors of the fragment by definition do not have a close parenthesis until after #FRAGMENT. If mentioned at all, prior siblings have both open and close parentheses before #FRAGMENT, and later siblings have both after. Thus, any element's attribute list ends at the first following (unquoted) parenthesis.

2.3.1 #PCDATA: Pseudo-elements

In mixed content, portions of character content between elements contribute siblings. In a fragment context specification that chooses to list siblings, such portions are specified by the keyword #PCDATA. This keyword may not have a repetition count or attributes.

2.3.2 #FRAGMENT: The fragment element

The token #FRAGMENT must be included at the point in the context where the fragment fits. This keyword may not have a repetition count or attributes.

2.3.3 #NET: NET-enabling start tags

The parameter "#NET" must be specified if and only if SHORTTAG is YES and the element for which it is specified is an ancestor that was opened with a NET-enabling start tag. It is necessary in this case so that the recipient can know to recognize a NET delimiter in the fragment. For example:

```
<chap/<sec/<p>Some text.....</p>//
```

The fragment context specification for the P element would then include:

```
CHAP #NET ( SEC #NET ( #FRAGMENT))
```

This parameter may also be specified for siblings which started with NET-enabling start tags, but this is unnecessary.

2.3.4 #MAP: Short reference maps

The parameter `#MAP=mapname` must be specified for any ancestor element that has a USEMAP declaration directly within it which precedes the fragment being sent, unless a nearer ancestor or the fragment itself overrides that map (making it inapplicable to the fragment). It is never needed in documents that do not use short references or that do not use USEMAP declarations within the document instance. For example:

```
<chap>
  <sec n=1>...</sec>
  <!USEMAP map37>
  <sec n=2>...</sec>
  <sec n=3>...</sec>
</chap>
```

The keyword must specify the name of the applicable map, for example `#MAP="map37"`. If more than one USEMAP has occurred, the most recent one must be specified, since it is the one in effect at the start of the fragment.

This parameter is permitted (but entirely unnecessary) for specifying short reference maps that are associated with all instances of an element type via a USEMAP declaration in the DTD. The recipient's parser already knows about those by virtue of the DTD plus the list of open element types. #MAP may also be specified for other elements described in the fragment context specification that contain USEMAP declarations, but this is also unnecessary.

2.4 Supplemental information

The preceding information is sufficient to enable a recipient to parse the fragment correctly; however, some additional information is commonly useful for application-specific processing of various kinds, and this resolution provides an optional way to send it. This resolution does not specify a method for senders and recipients to negotiate whether such information is sent. This resolution does, however, require that all recipient software be able to receive all optional information safely (even if it does not use it). It also provides, via the LEVEL item, a way for senders to inform recipients of what optional information they have actually sent.

2.4.1 Attributes

Processing specifications often test attributes to decide what to do, and may pass ancestor's attribute values downward to descendant elements. For example, setting SECURE=SECRET on a SECTION element might cause all elements within the SECTION to be hidden even though they do not themselves specify the SECURE attribute at all.

This resolution permits sending attribute lists for all elements for which GIs can be sent. Attribute values appear after the GI and are separated by white space. This is similar to the syntax of

SGML attribute specification lists. The syntax details for attribute values on CONTEXT items are exactly the same as specified above for the CURRENT item. For example:

```
(CONTEXT
  BOOK TYPE="MONOGRAPH" (
    BDY SECURE="PUBLIC" TOC="TRUE" (
      CHP #NET #MAP="map37" CNUM="1" (
        #FRAGMENT ))))
```

An element specification may provide no, some, or all of the attributes that the corresponding element instance had. Putting two assignments for the same attribute name with the same element is an error, and the correct error recovery is that the last assignment takes effect.

2.4.2 Siblings

Many auto-numbering methods use the sequence number of an element instance among its siblings, or more generally the number among just those siblings that fit some special criterion. For example, a section may be "3.2" because it is the second SEC within its parent CHP, while that parent CHP is the third CHP within the parent BDY. Because of this common need, this resolution permits listing the element types of siblings of the fragment element(s) and of each of its (their) ancestors.

For example, here the fragment is the fifth subelement of BDY (such as chapter 4), which is the first subelement of the root element BOOK (as in a document with no front matter):

```
(CONTEXT
  BOOK( BDY( INTRO() CHP() CHP() CHP() #FRAGMENT
)))
```

In addition, the attribute specification lists of those elements may be specified exactly as defined above for attribute lists of direct-line ancestors. A fragment context specification that provides attributes for ancestors is not required to send them for siblings as well. For example:

```
(CONTEXT
  BOOK TYPE="MONOGRAPH" (
    BDY SECURE="PUBLIC" TOC="TRUE" (
      INTRO() CHP() CHP() CHP() #FRAGMENT )))
```

A portion of character data in mixed content counts as a sibling. Such portions are specified by the keyword `#PCDATA` as shown here, which permits no associated attributes or parentheses:

```
(CONTEXT
  BOOK(
    BDY(
      INTRO() #PCDATA CHP() #PCDATA
      CHP() CHP() #FRAGMENT )))
```

2.4.3 Series of like siblings

A list of preceding siblings of a fragment element or an ancestor might contain a long sequence of repeated instances of the same element type. A repetition factor may be specified for any sibling GI listed in the fragment context specification. This optimization can provide great bandwidth benefits if a sender chooses to include sibling information at all.

A repetition count shall be specified by a separate token following the GI to which it applies, preceding any attributes, #NET, or #MAP. The token shall consist of "#" plus an unsigned decimal integer. It is an error to specify a repetition count of zero, and the correct error recovery is to ignore that elemspec. A repetition count of 1 is unnecessary but permitted.

For example, the specifications shown below are all equivalent:

```
(CONTEXT BOOK( BDY( CHP() CHP() CHP() CHP( P() P() #FRAGMENT ))))
(CONTEXT BOOK( BDY( CHP #2() CHP() CHP( P() P() #FRAGMENT ))))
(CONTEXT BOOK( BDY( CHP #4( P #2() #FRAGMENT ))))
```

If an element specification with a repetition factor is not closed before #FRAGMENT, then the last repetition is an ancestor of the fragment, and the other repetitions constitute prior siblings of that ancestor.

If an element specification gives both a repetition count and attributes, the specified attributes must have the same value for all element instances so combined (attributes not specified need not have uniform values). For example, a specification such as this states that all three chapters, the last one of which is an ancestor, have attribute TYPE=X:

```
(CONTEXT BOOK( BDY( CHP #3 TYPE="X" ( P( #FRAGMENT )))))
```

It may be useful in such cases to collapse runs of elements that share both element type and attribute values, but not combine potentially longer runs that share element type but not attribute values.

Note: the specification of an attribute with declared value ID on an element specification (elemspec) with a repetition factor greater than 1 would necessarily produce an invalid context (one in which multiple elements have the same ID).

3 Packaging the fragment and its fragment context specification

This resolution recognizes that there are various uses of SGML fragments and fragment context specifications. In particular, a fragment body need not be permanently associated with a specific fragment context specification, nor does this Resolution limit in any way whether a fragment body is associated with zero, one, or more fragment context specifications. Furthermore, this Resolution does not limit how a fragment body and its associated fragment context specification(s), if any, shall be associated. It is left to the individual applications, tools, and users to determine the most effective way given the particular circumstances. The principle goal of this Resolution is to define the fragment context specification language independent of any packaging issues.

However, this Resolution does realize that it will often be a practical necessity to "package" a fragment body and its associated fragment context specification; therefore, the following sections describe two possible ways to associate fragment bodies and fragment context specifications. Furthermore, for an implementation to be compliant with this Resolution, it must be able to process fragment entities packaged as described in the following section, though this in no way constrains users or applications to using this particular packaging method.

3.1 Embedding the fragment context specification in the fragment entity

When the concrete syntax of the fragment body uses the Reference Concrete Syntax values for the "processing instruction open" (PIO) and "processing instruction close" (PIC) delimiters, the entire fragment context specification can be embedded at the top of a fragment entity by making the fragment context specification string the content of one or more special SGML processing instructions (PIs) as described below.

The PI used to embed a fragment context specification at the top of a fragment entity must begin with the string `SO FRAG` followed by one or more whitespace characters (except for the special case of the `SO ESCPIC` PI described below). The content (that is, all system data between the PI's open and close delimiters except for `SO FRAG` and the immediately following whitespace) of the PI is taken as the fragment context specification.

If desired (for readability or to avoid exceeding certain quantities such as PILEN), the fragment context specification string can be split among multiple consecutive `SO FRAG` PIs. The content of all such PIs that occur prior to the fragment body are concatenated in order to produce the fragment context specification. (Note that, since the whitespace immediately following the initial `SO FRAG` characters will not be considered content of the PI when concatenating to reconstitute the fragment context specification, care must be taken when splitting the fragment context specification so that there is whitespace immediately following the split.) The fragment is deemed to begin at the first construct which is not a comment declaration, an SO FRAG or SO ESCPIC processing instruction, or whitespace.

When fragment context specifications are placed in PIs, they must not contain any instance of the "processing instruction close" (PIC) delimiter (e.g., ">" in the Reference Concrete Syntax). Should the need arise to encode the PIC delimiter—for example within a quoted attribute value specified for some ancestor or sibling—it is to be done as follows:

1. The `SO FRAG` PI that contains the character before the PIC delimiter shall be terminated after that character (with a PIC delimiter).
2. An `SO ESCPIC` processing instruction (e.g., `<?SO ESCPIC>` using the Reference Concrete Syntax PIO and PIC delimiters) shall follow, possibly separated by whitespace. (If there are consecutive occurrences of the PIC delimiter, multiple `SO ESCPIC` PIs shall be used.)
3. Another `SO FRAG` processing instruction shall follow, possibly separated by whitespace. This continues the fragment context specification starting immediately after the occurrence of the PIC delimiter(s) in the fragment context specification represented by the preceding `SO ESCPIC` PI(s).

The fragment context specification shall be reconstructed by concatenating all SO FRAG and SO ESCPIC processing instructions, but replacing each SO ESCPIC PI by an instance of the PIC delimiter.

Note: Most cases requiring the PIC to be embedded in the fragment context specification will arise within quoted attribute values, which means that quotation marks within *individual* SO FRAG PIs will not balance. This is not an error.

In the following example fragment entity, the “bdy” element’s “code” attribute has the value “>”:

```
<?SO FRAG
(DOCTYPE PUBLIC "-//Acme//DTD Book//EN")
(SOURCE SYSTEM "http://xyz.com/books/draft/b.sgm" (TREELOC 1 2 4))
(CONTEXT book ( bdy code=">
<?SO ESCPIC>
<?SO FRAG " date="1996-09-05" ( #fragment )))
>
<chp><ct>4: Printing</ct>
...
```

3.2 Multipart packaging protocols

Alternatively, the fragment body and its fragment context specification can be packaged using any protocol that permits including more than one storage object in an interchange package. A few examples of such protocols are tar, pkzip, stuffit, SDIF, and MIME Multipart/Mixed. In such a method, there are no constraints on characters within the fragment context specification (such as with the PIC in the previous section) unless they are imposed by the particular method chosen.

For example, the following example shows packaging a fragment body and its fragment context specification using MIME Multipart/Mixed:

```
Content-Type: Multipart/Mixed Boundary=fragment-example
--fragment-example

Content-Type: Application/X-SGML-Open-Frag-Spec
Content-Id: fragment.sof.960209.153601.123
```

```
 (DOCTYPE book PUBLIC "-//Acme//DTD Book//EN")
 (SUBSET SYSTEM "c:\foo.ent")
 (SOURCE SYSTEM "http://xyz.com/books/draft/b.sgm" (treeloc 1 2 4))
 (CONTEXT book ( bdy ( #fragment )))

--fragment-example

Content-Type: APPLICATION/SGML
Content-Id: fragment.sgm.960209.153602.345

<chp><ct>4: Printing</ct>
...
```

If sent as a separate file, the fragment context specification should be assigned the name "`fragspec`" and the extension "`.sof`" (for "SGML Open Fragment"). If an application associates a fragment context specification with a fragment body via an SGML Open Entity Catalog (TR9401), it shall do it via an extension whose keyword is `FRAGSPEC` and which takes as arguments two quoted storage object identifiers: that of the fragment context specification and then that of the fragment body.

If the Document Type Declaration is placed in the fragment entity just prior to the fragment body (so that the DOCTYPE item specifies `WITHFRAGMENT` instead of an external identifier), then the resulting combined storage object cannot be usefully referenced as an SGML text entity from within another document. If, on the other hand, the Document Type Declaration is separate, it may either accompany the fragment body and fragment context specification for transmission or may be omitted and then obtained by the recipient on demand using the external identifiers given in the fragment context specification's DOCTYPE and SUBSET items.

3.3 Additional examples

The following examples are intended to help further illustrate how this Technical Resolution might be applied.

```
<?SO FRAG
(DOCTYPE WITHFRAGMENT)
(CONTEXT book(front()body(chapter #2 chapter(section #4()#fragment))))
>
<!DOCTYPE book PUBLIC "-//Acme//DTD Acme Book//EN" [
<!-- This is the internal subset -->
<!ENTITY foo "bar">
]>
<section>
<!-- the section contents -->
</section>
```

By taking advantage of the defaults for DOCTYPE, the above "`(DOCTYPE WITHFRAGMENT)`" item can be omitted and the example can be written:

```
<?SO FRAG (CONTEXT book(front()body(chapter #2 chapter(section #4()#fragment))))>
<!DOCTYPE book PUBLIC "-//Acme//DTD Acme Book//EN" [
<!-- This is the internal subset -->
<!ENTITY foo "bar">
]>
<section>
<!-- the section contents -->
</section>
```