Davenport Group Draft Advisory Standard (DGDAS)

Editor: Steven R. Newcomb*

Version 1, 30 January 1992

Contents

1 Introduction **2**

2 References and Resources **4**

3 How to Read This DGDAS **5**

3.1 Editorial Conventions 5

3.2 Meta-DTD Conventions 6

3.2.1 Attribute List Conventions 6

3.2.2 Content Model Conventions 6

4 Davenport Architectural Forms **7**

4.1 On-site Location (`siteloc`) 7

4.2 Document Location Handle (`lochand`) 8

4.3 Location Signifier (`locsign`) 11

4.4 Index Link (`idxagg`) 12

*This version incorporates editing instructions given at the Davenport Workshop held 26-30 January, 1992, at Open Systems Solutions, Inc., Emeryville, California. Representatives of AT&T Unix System Laboratories, Fulcrum Technologies, HaL Computer Systems, Hewlett-Packard, IBM, Novell, O'Reilly & Associates, Open Software Foundation (OSF), Open Systems Solutions, Santa Cruz Operation (SCO), Silicon Graphics, Sirius Technologies, Technical Arts Group, and TechnoTeacher participated in the Workshop. For more information, contact Dale Dougherty, O'Reilly & Associates, Inc., 632 Petaluma Avenue, Sebastopol, California 95472. Tel. +1 800 338 6887 (outside California), +1 800 533 6887 (within California). Fax +1 707 829 0104. UUCP: uunet!ora!dale. Internet: dale@ora.com.

5 HyTime Support Declarations **14**
5.1 HyTime Base Module 14
5.2 HyTime Document Locations Module 15
5.3 HyTime Hyperlinks Module 17

6 Davenport Declarations **18**
6.1 Davenport Base Module 20

List of Figures

1 *Portions of a hypothetical Davenport-compliant document. These portions are identical to the portions given as examples in the text of this DGDAS.* 20

1 Introduction

This Davenport Group Draft Advisory Standard (DGDAS), dated 30 January 1992, defines a model and language for the representation of online and printed technical documentation – "man pages" – of the kind usually supplied by Unix vendors to their customers. The DGDAS will allow such vendors to "bundle" (to sell along with their systems) technical documents provided by a variety of publishers, and to give their customers access to all this information via user interfaces that they may define independently. The DGDAS will allow documents of dissimilar SGML document types[1] to be delivered as a single compound document with whatever aggregated indexes, tables of contents, cross-references, and webs of hyperlinks the systems integrators may wish to generate, without any special assistance from the originators and/or publishers. Document originators and publishers will be able to create and maintain a single version of each document without regard to the modes and styles of its ultimate delivery to users, since the DGDAS makes it possible for the same original document to be formatted for online delivery as hypermedia as well as for printing on paper.

This DGDAS is being developed in response to the widespread sentiment among software and systems vendors, and those who supply and maintain their documentation, that a standard representation of hypertext links – hyperlinks – should be

[1] SGML is the International Standard Generalized Markup Language defined by ISO/IEC 8879-1986. An SGML document type definition (DTD) describes the structure of a class of documents, all of which have the same allowable structural relationships between their various logical components ("elements"), and all of which share the same lists of properties ("attributes") for each type of element.

adopted by all, and that standardizing such a representation will benefit everyone involved. The reasons for this broad consensus include:

- the increased usefulness of information that can be accessed in the context of other documents increases the value of that information in the marketplace, and
- the increased usefulness of information that can access a broad base of other documents by hyperlink improves its own marketability and its publisher's profits, even when other publishers may receive additional revenue when such links are traversed by users.

This DGDAS consists of a set of templates, called "SGML Architectural Forms," to be used by document architects as they create the SGML document type definitions which establish the syntax of their documents. Because every element type or attribute defined in accordance with one of these forms takes on the semantic of the form, it is possible to build standard "Davenport Engines" — reusable generic software modules that perform the processing that is associated with the semantics of each of the Davenport architectural forms.

Some of the DGDAS architectural forms are based on, and are entirely consistent with, HyTime[2] architectural forms. When performing the semantic processing

[2]HyTime is the Draft International Standard Hypermedia/Time-based Structuring Language defined by ISO/IEC DIS 10744.

Aside from hyperlinks and facilities for addressing information objects (facilities that support hyperlinks), other candidates for standardization in Davenport-compliant documents include:

- representation of publishers' policies with regard to links to their documents and wholesale inclusions of portions of their documents in documents published by others. This would facilitate royalty accounting, and it would provide the infrastructure for notifying the authors of referencing documents when the referenced documents have been changed. A basis for standardizing the representation of these kinds of information already exists in HyTime's "activity tracking" facility.
- representation of schedules of events in time and space, using HyTime "Finite Coordinate Spaces (FCSs)." These representation facilities allow the presentation of information in time and space to be specified independently of any particular platform. Among other things, HyTime's FCS facilities allow explicit distinctions to be made between formatting information that is essential to the meaning of information that constitutes a table, for example, and formatting information which is purely renditional and which can safely be ignored by applications that have no need to format a table in order to make use of tabular information. FCS "schedules" can also be used to represent time-based animations, simulations, and multimedia presentations in such a way that documents that incorporate multimedia information objects can be created and maintained in a single source form which can be formatted for use on a wide variety of systems.

required by the DGDAS on elements which conform to HyTime architectural forms, the behavior of a Davenport Engine will be indistinguishable from the behavior of a HyTime Engine acting on the same data, except that the Davenport Engine will apply additional constraints in accordance with the prescriptions of particular Davenport architectural forms. Therefore, a Davenport Engine need not duplicate the functionality of a HyTime Engine, but could instead consist of a set of validating routines added as a module to a system that already contains the appropriate subset of the functionalities of a HyTime Engine.[3]

Like the HyTime Draft International Standard on which it is based, this Davenport Group Draft Advisory Standard is intended to be used as an infrastructure for platform-independent information interchange; it is *not* intended for encoding the internal representation of information on which specific hypertext application programs act. Developers of document processing applications will provide facilities for importing and exporting documents in such a way as to conform to Davenport constructs, but the manner in which a document is represented within an application is defined neither in HyTime nor by Davenport.

2 References and Resources

1. ISO 8879, *Standard Generalized Markup Language* (SGML). This is the international standard metalanguage on which HyTime and DGDAS are built.

2. ISO 9069, *SGML Document Interchange Format* (SDIF). This standard shows how documents represented in SGML, including HyTime and DGDAS documents, can be interchanged between similar and dissimilar systems in a fashion congruent with the Open Systems Interconnect (OSI) seven-layer model, employing Abstract Syntax Notation 1 ("ASN.1").

3. ISO 9070, *Registration Procedures for Public Text Owner Identifiers.* This standard provides a procedure whereby SGML text such as DTDs can become publicly available by reference to an assigned unique internationally

[3]Some of the features of HyTime, such as the ability to schedule events in virtual or real time and/or space, are not presently used in this draft. This presents no problem, because HyTime's modular design allows applications and documents to support the various features of HyTime selectively. However, in the likely event that online technical documentation will eventually include such things as schedulable multimedia objects, the decision to become HyTime-compatible at this early stage offers the advantage that the structure of today's text and graphic information will be 100% compatible with that of tomorrow's multimedia documents.

standardized number. The DGDAS may ultimately become "public text" by the method outlined in ISO 9070.

4. ISO DIS 10744, *Hypermedia Time-based Structuring Language* (HyTime). Itself founded on SGML, HyTime is the standard foundation on which the DGDAS is built. (See footnote 2.)

5. The *SGML SIGhyper Newsletter* is published by the SGML Users' Group Special Interest Group on Hypertext and Multimedia. It publishes examples of HyTime documents and other information relevant to the DGDAS, including copies of the HyTime DIS itself. Contact Chris W. Higgins, Treasurer, SGML SIGhyper, c/o TechnoTeacher, Inc., 1810 High Road, Tallahassee, Florida 32303 USA; tel: +1 904 422 3574; fax: +1 904 386 2562; Internet: chris@cmr.fsu.edu.

3 How to Read This DGDAS

This DGDAS is expressed using a formalism that was first used to represent the HyTime DIS: the so-called "meta-DTD." Rather than containing the `ELEMENT` and `ATTLIST` declarations normally found in SGML document type definitions (DTDs), this DGDAS sets forth templates, or "SGML architectural forms," which are intended solely for human readers, and which are not meaningful to SGML parsers. SGML architectural forms are, in essence, rules for constructing element types, including attribute definitions and content model constraints. The grammar of SGML architectural forms resembles the grammar usually used to express SGML document type definitions; the similarities make it convenient for document architects to use the architectural forms as templates as they create specific Davenport-compliant DTDs. (Such DTDs will normally include both Davenport and non-Davenport components.)

3.1 Editorial Conventions

Section 4 consists of the Davenport Architectural Forms (the SGML templates) of this DGDAS. Each architectural form is presented in three steps:

1. The architectural form is defined using the formalism developed to express the HyTime meta-DTD; this is followed by a description of the form in English.[4]

[4] For a thorough understanding of each Davenport architectural form that is based on a HyTime architectural form, it is imperative to consult the HyTime DIS. This DGDAS is not meant to recapitulate the HyTime DIS.

2. An example of an SGML `ELEMENT` and `ATTLIST` declaration that conforms to the architectural form is given and explained. The example appears just as it might appear in some actual Davenport-compliant DTD; it addresses some particular purpose, but there is no implication intended that that purpose is the only possible one (far from it!).

3. Finally, an instance of the exemplary `ELEMENT` and `ATTLIST` is given and explained. These instances are contrived in such a way that they all work together as if they appear in a single hypothetical document (see Figure 1 at the end of this DGDAS).

3.2 Meta-DTD Conventions

As in the HyTime DIS, architectural forms are specified using element type forms and attribute list type forms.[5] Certain conventions have been followed in these declarations to clarify the element content models and the attribute definitions and their data types. The keywords `element` and `attlist` are written in lower case in order to emphasize the fact that this is a meta-DTD rather than a true SGML DTD.

3.2.1 Attribute List Conventions

Components contributed by HyTime architectural forms to Davenport architectural forms are identified by associated comments. Constraints on attribute values and data types imposed by HyTime or by this DGDAS are also indicated by comments.

Common attributes are included in the formal specification of a particular architectural form only when they interact with other architectural forms. For example, an `ID` attribute will be included for a form only when some architectural form includes an attribute that references it. As with HyTime, DTD designers are always free to add application-specific attributes *ad libitum* to Davenport-compliant element declarations.

3.2.2 Content Model Conventions

Content models for Davenport architectural forms are shown as ordinary SGML content model declarations. The content models may be further defined by associated comments, which may indicate additional lexical and/or semantic constraints on the content of any actual conforming element type. Any actual element type's content

[5] This DGDAS does not define any Davenport attribute list type architectural forms, but future versions may do so.

model may be *more* constrained than the model as described and constrained by the architectural form, but it may not be *less* constrained. Some examples:

- When an architectural form's content model contains a user-definable parameter entity reference, a conforming element type may be considered "wide open" with respect to the element types, etc., that it may contain, within the constraints (if any) on that parameter entity and on the content model itself.
- Similarly, the content model `ANY` in an architectural form means that a conforming element type may contain anything. However, in either of the foregoing cases, an actual DTD may constrain the content model of a given element type to a particular element structure and/or data type, since it would then be more (and not less) constrained than that prescribed by the architectural form.
- The content model `EMPTY` in an architectural form would mean that a conforming element type may not contain anything; it is already fully constrained.

4 Davenport Architectural Forms

In addition to the architectural forms defined in this document, this DGDAS uses the HyTime document location addressing and linking constructs; Davenport applications will need to provide support for much of the HyTime Location Address Module and the HyTime Hyperlinks Module (see Section 5 and the HyTime DIS for details).

4.1 On-site Location (`siteloc`)

```
<!element siteloc - - ANY -- A potential anchor
                             marked in context. -->
<!attlist siteloc
    Davnport    NAME    #FIXED        siteloc
    id          ID      #REQUIRED
>
```

Davenport defines an **on-site location** element type form, `siteloc`, which provides a Davenport-standard way to point to fragments of content consistent with the common practice of marking places in a manuscript to which an index entry, for example, will refer. A `siteloc`-form element would normally be used to apply a unique identifier to something which would not otherwise be an element, and which

therefore could not have a unique identifier. The content of a siteloc must be valid for the context in which the siteloc appears; a validating Davenport engine must check whether this is so.

Sample declarations using the `siteloc` form:

```
<!ELEMENT partname - - (#PCDATA) -- A name of a hardware
                                    component. -->
<!ATTLIST partname
    Davnport    NAME    #FIXED      siteloc
    id          ID      #REQUIRED
>

<!ELEMENT indexme - - (#PCDATA) -- An item in the
                                   context of #PCDATA
                                   to be indexed. -->
<!ATTLIST indexme
    Davnport    NAME    #FIXED      siteloc
    id          ID      #REQUIRED
>
```

Sample instances of `siteloc` element types:

```
<para id=p1>When disassembling the
<partname id=part1>instrument bezel</partname>,
be sure to save the <indexme id=ixme1>grommet</indexme>
for later <indexme id=ixme2>recycling</indexme>.</para>
```

It is possible to refer to the exact location of the words "instrument bezel," "grommet," and "recycling" by means of the unique identifiers of their `siteloc`-type elements (`part1`, `ixme1`, and `ixme2`, respectively). These locations could be used as hypertext buttons, index entries, anchors, etc. It is also possible to refer to the whole paragraph in the same way, using its unique identifier (`p1`).

4.2 Document Location Handle (`lochand`)

```
<!element lochand - O (#PCDATA)
    -- lextype(idrl), reftype (ANY) --
    -- HyTime constraint on content model: must be
```

```
      list of valid local IDs; IDs may resolve to
      any element type.
   -- Content is anchor location: if empty, HyTime link
      element content is anchor; if IDREFs, anchor is
      aggloc -->
<!attlist lochand
   Davnport    NAME    #FIXED      lochand
   -- HyTime linkend attributes (see also HyTime DIS) --
   HyTime      NAME    #FIXED      linkend
   id          ID      #REQUIRED
   traverse -- begin(B), return(R), both(BR) --
               (b|r|br) br
   aggacc   -- Aggloc access: direct(D), indirect(I)
               cannot be an aggregate(N) --
               (d|i|n)  n
   endterm  -- Link end term information --
               IDREF    #IMPLIED
            -- Default: use anchor --
            -- Davenport-added constraint on
               endterm: ID must be element conforming
               to Davenport locsign form.
   -- HyTime aggloc attributes (required if aggregate
      linkends are allowed by aggacc attribute's value) --
   ordering -- Is ordering of locations significant? --
               (ordered|noorder) noorder
   maxlevel -- Level that IDREF resolution cannot exceed,
               relative to this aggloc.  Default: all
               levels --
            -- Resolution includes query evaluation --
            -- Must be a signed non-zero integer --
               NMTOKEN  "-1" -- Default: all levels --
   merged   -- Should levels (after resolution to maxlevel)
               be merged into one level, or stay nested? --
               (merged|nested) nested
   aggsrc   -- Aggregate source: aggregate or members.
               i.e.,  should members be treated separately
               or be concatenated? --
               (agg|members) members
>
```

The HyTime **link end definition** (`linkend`) architectural form, on which the Davenport **document location handle** (`lochand`) architectural form is built, is designed to associate a variety of information with document locations. The generic identifier of a `linkend` element type may apply an application-specific semantic indicating why the document location is being pointed at. In addition, `linkend` element types may point to information that will be shown to a user indicating that the linkend is available for traversal; this information is called the "linkend term" or "`endterm`" information in HyTime jargon. Finally, several other pieces of information are also given with each `linkend` type element regarding the rules governing traversal to or from this `linkend`, and the intended treatment of any aggregate information in cases where the `linkend` gives access to multiple anchors.

An author can provide `lochand`-type elements within a document to provide conveniently encapsulated "handles" for other documents to link to.

This DGDAS `lochand` architectural form is in all respects identical to the HyTime `linkend` architectural form, except that the `endterm` attribute's value must be the unique identifier of an element conforming to the Davenport "`locsign`" architectural form; a validating Davenport application must check whether this is so.

Sample element type declaration using the `lochand` form:

```
<!ELEMENT idxhand - - (#PCDATA)
    -- HyTime constraint: must be a list of valid
       local identifiers (IDs).
    -- Link end handle of something that is pointed
       to  for the purpose of associating it with
       its index information (see idxentry). -->
<!ATTLIST idxhand
    Davnport    NAME    #FIXED      lochand
    HyTime      NAME    #FIXED      linkend
    id          ID      #REQUIRED
    traverse    NAME    #FIXED       br
        -- User can traverse to and from this linkend --
    aggacc      NAME    #FIXED       n
        -- Cannot refer to an aggregate location --
    endterm     IDREF   #IMPLIED -- default: use anchor --
        -- Davenport-added constraint: must point
           directly at an element that conforms to
           the locsign architectural form). --
```

```
        -- (HyTime aggloc attributes are not needed for
           idxhand because aggacc = n.) --
>
```

Sample instance of `lochand`-form element type:

```
<idxhand id=ixhand1 endterm=ixentr1>ixme2 ixme3</idxhand>
```

This instance associates the `endterm` (Davenport: `locsign`) information contained in the element whose unique identifier is `ixentr1` with the document locations whose unique identifiers are `ixme2` and `ixme3`. The entire assembly can now be referred to (i.e., hyperlinked) via its own unique identifier, `ixhand1`, both from within the document and from other documents.

4.3 Location Signifier (`locsign`)

```
<!element locsign - - ANY -- signifier (hypertext button,
                              index or toc entry, etc.) -- >
<!attlist locsign
    Davnport    NAME    #FIXED      locsign
    id          ID      #REQUIRED
>
```

HyTime does not define any particular structure (i.e., architectural form) for elements that contain information intended to be shown to the users of hyperdocuments to indicate that it is possible to traverse to a particular document location that serves as an "anchor." This DGDAS declares a Davenport architectural form for this purpose, **location signifier** (`locsign`), whose content model is left to the discretion of the DTD designer. The purposes of `locsign` are merely to identify element types that are intended to serve as endterms to Davenport engines for document validation purposes, and to require a unique identifier attribute called "`id`."

Sample element-type declaration using the `locsign` form:

```
<!ELEMENT idxentry - O (#PCDATA)
        -- Index Entry (idxentry).  A word or phrase
           that will be displayed (or printed) in an
           index or on a hypertext button in order to
           associate linkends with it, or, in the case
```

```
          of a printed index, to give page
          references. -->
<!ATTLIST idxentry
    Davnport    NAME    #FIXED      locsign
    id          ID      #REQUIRED
    butgraph    ID      #IMPLIED
        -- id of element containing graphic information
           about the appearance of the hypertext button
           on which the #PCDATA content will appear -->
```

Sample instance of `locsign`-form element type:

```
<idxentry id=idxentr1 butgraph=ic1>recycle</idxentry>
...
<icon id=ic1>... (graphic info here) ...</icon>
```

According to the semantics of this particular hypothetical DTD and its associated set of applications, the word "recycle" will appear in an index, if printed, and on a hypertext button, if it appears in an interactive displayed index in an online document. In the case of the online document, the button icon described in the element whose unique identifier is `ic1` will be used, and the word "recycle" will appear on that button.

4.4 Index Link (`idxagg`)

```
<!element idxagg    - O      (#PCDATA)
    -- HyTime constraint: content must be a list
       of valid local unique identifiers (IDs). --
    -- Davenport constraint: IDs must refer to
       lochand-form elements. -->
<!attlist idxagg
    Davnport    NAME    #FIXED      idxagg
    -- HyTime agglink attributes --
    HyTime      NAME    #FIXED      agglink
    id          ID      #REQUIRED
    -- aggloc attributes --
    ordering -- Is ordering of locations significant? --
                (ordered|noorder) ordered
    maxlevel -- Level that IDREF resolution cannot exceed,
```

```
                relative to this aggloc.  Default: all
                levels --
             -- Resolution includes query evaluation --
             -- Must be a signed non-zero integer --
                NMTOKEN  "-1" -- Default: all levels --
    merged   -- Should levels (after resolution to maxlevel)
                be merged into one level, or stay nested? --
                (merged|nested) nested
    aggsrc   -- Aggregate source: aggregate or members.
                i.e.,  should members be treated separately
                or be concatenated? --
                (agg|members) members
>
```

The Davenport **index link** (idxagg) architectural form has the semantics of an index, i.e., an ordered list of indexed items intended to help the user of a document locate things of interest. It is a single, many-ended link whose linkend elements must conform to the lochand Davenport architectural form, and which in all other respects conforms to the HyTime agglink architectural form. Because it is a HyTime agglink, a validating HyTime or Davenport application must check whether the content consists entirely of locally valid unique identifiers; because it is a Davenport idxagg, a validating Davenport application must check whether all of the unique identifiers in the content ultimately refer (perhaps via one or more HyTime document location pointer elements) to Davenport lochand type elements.

Sample element type declaration using idxagg form:

```
<!ELEMENT index - - (#PCDATA)
    -- must contain only valid unique identifiers
       of lochand-conforming elements.  -->
<!ATTLIST index
    Davnport    NAME    #FIXED      idxagg
    HyTime      NAME    #FIXED      agglink
    id          ID      #REQUIRED
    ordering    NAME    #FIXED      ordered
        -- order is significant --
    maxlevel    NMTOKEN #FIXED      "-1"
        -- Never abort location pointer resolution;
           resolve the whole chain all the way to
```

```
        the anchor. --
    merged      NAME    #FIXED      merged
        -- Always merge all levels into one level. --
    aggsrc      NAME    #FIXED      members
        -- Do not concatenate the resolved members
           of the aggregate.  -->
```

Sample instance of idxagg-form element type:

```
<index id=myindex>ixhand1 ixhand2 ixhand3</index>
```

This `index` has only three entries in it. The `lochand`-form element whose unique identifier is `ixhand1` appears in an example above; all examples are shown in an articulated fashion in Figure 1 at the end of this DGDAS.

5 HyTime Support Declarations

The following module support declarations are required with the use of the indicated HyTime architectural forms and options (see the HyTime DIS for details).[6]

5.1 HyTime Base Module

```
<?HyTime base>
```

Always required for any document using HyTime, the HyTime Base Module[7] ("`base`") minimum configuration consists of:

- SGML itself, with its element, attribute, and entity object representation and identification facilities, document type definition capability, ability to include multimedia objects represented in any content notation, separation of processing specifications from intrinsic information, ability to associate multiple sets of processing specifications ("styles") with element types and individual elements, and its transparent entity structure.

[6] Additional HyTime support declarations may be needed when more HyTime architectural forms are incorporated by the DGDAS.

[7] The base module has several optional facilities which are not used in this DGDAS, but which may be included in future versions. These optional facilities include `lextype`, `hylex`, `eltype`, `reftype`, `dvlist`, `desctxt`, and `activity`. See the HyTime DIS for details.

- the SGML Document Interchange Format defined in ISO 9069, which provides compatibility with Open Systems Interconnection (OSI) protocols.

- the SGML APPINFO parameter of an SGML document's "SGML declaration," which allows substitution of a string other than *HyTime* for use as the "HyTime Prefix." This facility allows a potential name conflict to be resolved without loss of information.

- the HyTime Support Declaration facility, which provides a means to express which features of HyTime must be supported in order to process the document. (The above "<`?HyTime base`>" is an instance of such a declaration.)

- the HyTime architectural form attribute name (`HyTime`), which identifies the HyTime architectural form of the element.

- the HyTime document `doc` architectural form, which represents the highest level element in a HyTime document element structure. Its content model establishes a flexible meta-structure for HyTime-conforming document elements.

- a facility for aliasing the names of attributes defined by HyTime (`HyNames`), so as to avoid name conflicts.

- a content location attribute (`conloc`), which identifies the content of the element when it is not in the syntactic content. In addition, a content location extraction attribute (`conlocx`) is provided, which indicates whether the element or data at the content location identified by the `conloc` attribute is to be used for the content, or whether the content of the element at the content location is extracted for use in the content of this element.

- a bit combination skip count attribute `bitskip`, which specifies the number of unparsed bit combinations that must be skipped after the start-tag to reach the first bit combination after the end of the syntactic content. The purpose of `bitskip` is to allow optimal performance by SGML parsing applications that can benefit from knowing how big an information object is, just so it can be ignored efficiently.

5.2 HyTime Document Locations Module

```
<?HyTime locs aggloc anysgml anydtd exidrefs nmquery notsrc>
```

The HyTime Location Address Module ("`locs`") provides a variety of facilities for indicating the location of particular pieces of information (i.e., portions of documents).[8] The `locs` module has the following standard features:

- The HyTime **bibliographic location address** (`bibloc`) is a reference to an information object that applications are not expected to access automatically.
- The HyTime **named location address** (`nameloc`) element-type form associates a local unique identifier with one or more named elements or entities that are named in a HyTime **name list** (`namelist`) element. (The `namelist` HyTime architectural form is also a standard feature of the HyTime `locs` module.)
- The HyTime **location source attribute** (`locsrc`) of HyTime document location address elements identifies the source (perhaps the data contained in an element, or perhaps another location address element) from which the current location address element is intended to select some portion. The `locsrc` attribute allows "chains" of document location address elements to identify a portion of a document by a process of gradual refinement. The associated `evalsrc` attribute specifies whether the source identified by `locsrc` is intended to be evaluated as a further location address element, or is intended as the ultimate address.
- The HyTime **attribute location address** (`attloc`) addresses the value of an attribute of an object by the attribute name, or, if no attribute name is specified, the generic identifier of the object.

The `locs` module has the following optional features required by this DGDAS:

- The `aggloc` support declaration gives notice to the application that multiple locations may be aggregated and referenced by means of a single element. `Aggloc` requires the support of the HyTime Hyperlinks Module ("`links`"). The set of attributes associated with the `aggloc` support declaration allows policies to be expressed regarding issues that come up when aggregate locations can be pointed to via a single reference. These attributes are:

[8] Not all of the architectural forms provided in the HyTime Location Address Module are required for the support of documents conforming to this DGDAS. Several of the HyTime forms require the support of the HyTime Measurement Module and/or the HyTime Finite Coordinate Space Module, neither of which is required for Davenport documents.

  - `ordering`, which is used to specify whether or not the order of the locations must be preserved by the application;
  - `maxlevel`, which is used to specify the number of levels of IDREF resolution which may be allowed to occur;
  - `merged`, which is used to specify whether or not the information regarding which level of IDREF resolution was the terminal resolution should be preserved (`merged=nested`) or discarded (`merged=merged`) by the application; and
  - `aggsrc`, which is used to specify whether or not the resolved members of the aggregate should be concatenated (`aggsrc=agg`) or allowed to remain separate (`aggsrc=members`) by the application.

- The `anysgml` support declaration gives notice to the application that SGML entities may have to be parsed with respect to SGML declarations other than the one which governs the entity in which the reference is made (because, for example, they may conform to different concrete syntaxes).

- The `anydtd` support declaration gives notice to the application that SGML entities may have to be parsed with respect to document type definitions or link process definitions (DTDs or LPDs) other than the base.

- The `exidrefs` support declaration gives notice to the application that certain specified attributes may be allowed to refer to locations outside the current document.

- The `nmquery` support declaration gives notice to the application that elements conforming to the HyTime **name list query** (`nmquery`) architectural form may be used. These elements contain expressions in query languages which return lists of names of elements or entities based on the semantic properties of those elements or entities. (HyTime does not specify the nature of the query language(s).)

- The `notsrc` support declaration gives notice to the application that elements conforming to the HyTime **notation-specific location address** (`notloc`) architectural form may be used. These elements contain expressions in query languages which address a portion of data whose interpretation is subject to the rules of a data content notation. HyTime does not specify the nature of the query language(s).

5.3 HyTime Hyperlinks Module

`<?HyTime links agglink manyends>`

The HyTime Hyperlinks Module ("`links`") provides a variety of facilities for expressing the fact that two or more portions of information bear some relationship to one another. HyTime does not standardize the semantics of particular hyperlink types (i.e., particular purposes for expressing hyperlinks); it standardizes only the syntax required to indicate which information is intended to be considered the `linkend`s of a hyperlink, which information is intended to serve as the user's preview (`endterm`) of each `linkend` (i.e., how the user is informed that a hyperlink is available for traversal), and certain rules or policies regarding traversal.

The HyTime `links` module has the following standard features:

- The HyTime **independent link** (`ilink`) architectural form provides for the expression of the idea of linkage between any number of portions of information. The context of an `ilink` may or may not be significant, depending on the semantics of the application.

- The HyTime **contextual link** (`clink`) architectural form provides for the expression of links that have exactly two linkends, one of which is the context of the `clink` itself.

The optional features of the Hyperlinks Module required by this DGDAS include:

- The `agglink` support declaration, which gives notice to the application that elements conforming to the HyTime **aggregate location link** (`agglink`) architectural form may be used. These elements are used to turn a list of locations into an aggregate location; their content consists of any number of SGML unique identifiers, each of which points directly or indirectly (via a document location pointer chain) to some document location. `Agglink` differs from `ilink` in that `ilink`-form elements permit application-defined semantic distinctions to be made explicit via their attribute lists, which can point to a fixed number of link ends, while `agglink` allows an indefinite number of linkends that do not have a HyTime-defined facility for expressing distinctions between them.

- The `manyends` support declaration gives notice to the application that elements conforming to one or more of HyTime's hyperlink architectural forms may have more than two linkends.

6 Davenport Declarations

Since it relies on HyTime for its semantics, the DGDAS can afford to be simple; it need not have the complex array of modular support declarations that HyTime requires. For the present, anyway, it is only necessary that Davenport documents be recognized as conforming to the DGDAS. However, future versions of the DGDAS that support specialized forms of HyTime activity tracking and/or tabular data representation via HyTime FCSs may require more elaborate arrangements. Therefore, the DGDAS uses the same strategy for Davenport support declarations that is used in HyTime. This strategy gives future versions of the DGDAS the ability to add plenty of specialized architectural forms, while maintaining total backward compatibility and while providing for future modularity:[9]

- A document indicates conformance to the DGDAS by the APPINFO parameter of the SGML declaration, which specifies the name of the "Davenport Prefix" that identifies SGML processing instructions that represent "Davenport Declarations." Davenport declarations indicate the facilities required to support the document. The format is:

  ```
  Davnport=name
  ```

  if a prefix name is to be specified, otherwise "`Davnport`" alone. The prefix name must be a valid name in the concrete syntax of the SGML declaration.

- A Davenport declaration is represented as an SGML processing instruction that begins with the Davnport prefix, followed by a "declaration name" and other parameters as required by the declaration type. Each of the parameters must be preceded by one or more "white space" characters: i.e., in SGML terms, RS, RE, and SPACE. White space characters can also follow the last parameter. The case of the parameters is not significant.

- A Davenport document includes in its prolog one or more "Davenport support declarations" that collectively identify all of the modules and options for which support is required in the document. A Davenport support declaration is represented as an SGML processing instruction whose declaration name is "support." The name must be followed by one of the module names in the following list, thereby declaring that support is required for that module:

[9] These needs for future additions, backward compatibility, and modularity explain why there is, in this DGDAS at least, "so much dressing for so little salad."

`base`

The module name must be followed by the names of the module's optional facilities (if any) for which support is being declared. For example, using the SGML reference concrete syntax,

```
<?Davnport  SUPPORT  base  option1  option2>
```

(There are currently no optional facilities in the DGDAS.)

6.1 Davenport Base Module

The Davenport Base Module (`base`) requires support for all of the facilities described in Sections 4 and 5 of this DGDAS.

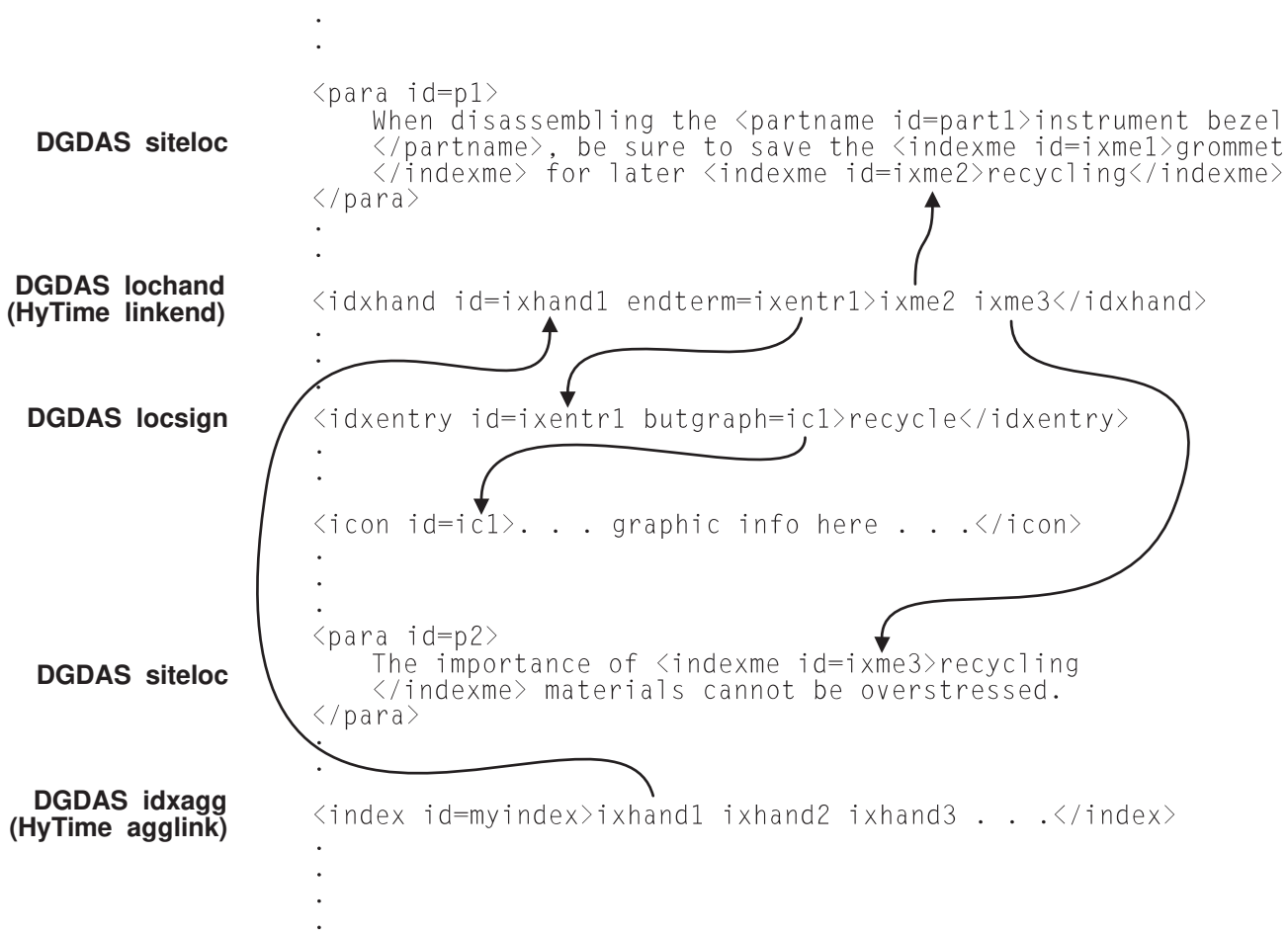

Figure 1: *Portions of a hypothetical Davenport-compliant document. These portions are identical to the portions given as examples in the text of this DGDAS.*