**PERUSE/XMP**

**TIMING RESULTS**

**IBM 3033**

| MODEL: | 10 periods | | 20 periods | |
|---|---|---|---|---|
| ROWS: | 254 | | 504 | |
| COLUMNS: | 475 | | 955 | |
| ELEMENTS: | 1802 | | 3622 | |
| UNIQUE ELEMENTS | 581 | | 1119 | |
| | WHIZARD | XMP | WHIZARD | XMP |
| STARTING BASIS | NO | NO | NO | NO |
| OPTIMIZATION TIME | | | | |
| CPU SECONDS | 3.4 | N/A | 8.3 | N/A |
| ELAPSED TIME, MM:SS | 00:53 | 01:00 | 01:57 | 04:45 |
| ELAPSED TIME DIFFERENCE | 7 seconds | | 2 min 48 sec | |
| % | BASE | +13 | BASE | +143 |

87



# SESSION REPORT

SHARE

61 | A216 | Considerations in Designing a GML Application | 75

SHARE NO. | SESSION NO. | SESSION TITLE | ATTENDANCE

Document Composition | Sharon Adler | BCG

PROJECT | SESSION CHAIRMAN | INST. CODE

Boeing Computer Services, 7970 Gallows Ct, Vienna, VA 22180 (703) 827-4629

SESSION CHAIRMAN'S COMPANY, ADDRESS, and PHONE NUMBER

**Considerations in Designing a GML Application**

Truly Donovan

IBM Corporation
555 Bailey Avenue
San Jose, California 95150

DOCUMENT COMPOSITION PROJECT

Session Number A216

*Abstract:*

This presentation focuses on the practical considerations in designing a major GML (Generalized Markup Language) application to support a wide range of users producing a number of different document types on a range of output devices. In addition to the near-term objective of producing formatted documents in a cost-effective way, the design must address potential long-term uses of the text that is being created today.

The speaker is a member of the team working on the design of IBM's internal GML application, and will use that design as the basis for discussion.

## Introduction

Starting in 1977, we formed an interdivisional committee, consisting primarily of publications personnel, to undertake the design of a common Generalized Markup Language (GML) application for use throughout the IBM internal publishing community — that is, the people who produce the user manuals and service manuals that support our products. Although our focus was, and continues to be, on the needs of the publishing arena, it was also apparent that our requirements represented a superset of many of the document processing needs of the corporation in general, and that we were in fact addressing a broader user group on a wider range of documents.

In 1977, however, manuals alone was a sufficient challenge.

The resulting design and implementation is known as "ISIL," for "Information Structure Identification Language," and that's how I will be referring to it from now on.

***DESIGN OBJECTIVES***

A COMMON SYSTEM:

- DOCUMENT PORTABILITY
- WIDE RANGE OF DOCUMENTS
- BROAD USER BASE
- USER PRODUCTIVITY
- REDUCED TRAINING
- REDUCED DEVELOPMENT INVESTMENT
- LONG DOCUMENT LIFE

There was nothing at all exotic about our objectives. First and foremost, we needed a common system — I'll return to the subject of commonality later.

Document portability was itself a sufficient motivation to undertake this effort. The lack of it had been a major source of frustration and wasted resource.



The other objectives grew in importance as we came to realize just how much was possible with the direction we had taken — that is to say, how much a good GML design was going to buy for us.

The documents we support cover everything from a half-page list of things to do to an 800-page complex manual. Our users range from the casual and occasional to the full-time publications production expert.

User productivity had to be considered as well, given the enormous volume we produce. And we have experienced a quantum jump in productivity. Similarly, we have reduced the user training requirement in two areas: ISIL is easier to learn and to use than anything we've had in the past, and because it is common, users don't have to be retrained when they move to a new environment.

Of course a common system means a single development and maintenance effort, and the benefits of that are obvious.

Finally, we needed a system that would carry us forward into the future. Many of our documents have a life expectancy — the time for which they are active and subject to revision — of perhaps ten years or more. (I was impressed by that until I learned that aircraft maintenance manuals have a life expectancy of 25 years.) Many of the books in my own organization have been through at least two "conversions" from one text processing system to another. I don't have to tell you how unsatisfactory an activity that is.

Let's return for a moment to the subject of commonality. If everyone used, say, native SCRIPT/VS formatting controls, that could be said to be a "common system." But that level of commonality is totally inadequate to our needs.

***A "COMMON" SYSTEM***

COMMON SOURCE FILE MARKUP

- INDEPENDENT OF SPECIFIC FORMATTING CONTROLS
- READABLE AND READILY UNDERSTANDABLE

- NOT NECESSARILY A COMMON OUTPUT APPEARANCE

We needed a common *source file markup*, but one that was independent of specific formatting controls. Specific formatting controls are just barely portable within a single organizational entity, and then only if you dictate strict markup protocols. They become distinctly unportable when you plug in a new output device, or just change your mind about how things should look on the output page on the old device.

Furthermore, portability does not mean simply the ability to process the document with some reasonable output result. It also means that someone else can read the source files and readily understand the markup (our source files pass through many hands in the course of their life). A string of formatting controls does not pass this test.

Understand that we were not particularly motivated by the desire to achieve a single output appearance across all documents. That is, of course, an objective that is readily met with a GML implementation. We were motivated instead to achieve multiple output appearances from a common markup. This requires a little more sophistication in your GML implementation.

In fact, in the six years since we began our work, the need to support dramatically different output appearances has grown substantially.

## Document Types

ISIL currently supports five document types:

***ISIL DOCUMENT TYPES***

- general documents (GDOC)
- user manuals (USERDOC)
- maintenance library part-number-controlled documents (MLPNDOC)
- maintenance library form-number-controlled documents (MLFNDOC)
- standards documents (STDDOC)

**general documents (GDOC)**
This is as defined by the Document Composition Facility (DCF) starter set. Although many of the tags we've added could be considered of a "general document" nature, we decided as a practical matter to limit the definition to the starter set.

The implementation does not police this definition, however. That is, a document defined as a GDOC (general document) can contain tags that are outside the general document definition, and ISIL processes them correctly without a murmur. We're not too concerned about this, because it is a trivial matter to convert a GDOC to a USERDOC, and we don't currently see any exposure in tolerating USERDOC tags in a GDOC. If we ever had to, we could police the definition and force the conversion.

**user manuals (USERDOC)**
This is the classification for user manuals for both hardware and software. It is the ISIL default document type. A document that does not otherwise declare itself is treated as a USERDOC.





It is also, in a sense, the ISIL "general document" or perhaps even "universal document." That is, users who are producing documents of a type that we do not explicitly support — for example, programming functional specifications — use the USERDOC definition.

**maintenance library part-number-controlled documents (MLPNDOC)**
This is one of two document types recognized for maintenance library manuals.

**maintenance library form-number-controlled documents (MLFNDOC)**
This is the other document type recognized for maintenance library manuals.

**standards documents (STDDOC)**
Standards documents do not represent a very large volume, as compared to some of the other things we do, but they are created at many different locations and require a common output appearance. This was a natural for "ISILization." Adding this document type involved creating only two new tags that are unique to the document type; they are formatted differently from USERDOCs, but use the same "core" tag vocabulary.

With respect to those document types that we do not explicitly support, when the formatting requirement differs at all from what ISIL normally produces for USERDOCs, the primary area of difference is the treatment of running headings and footings.

## Design Considerations

I'm going to discuss some general design considerations, in these areas:

***GENERAL DESIGN CONSIDERATIONS***

- HOW MANY TAGS?
- HOW MANY ATTRIBUTES?
- GENERIC OR SPECIALIZED?
- END TAGS?
- CONSISTENCY

I will follow that with a discussion of a few of the specific design problems and how we dealt with them.

### *How Many Tags is Too Many Tags?*

Having posed the question "how many tags is too many tags?," I don't propose to answer it.

***HOW MANY TAGS IS TOO MANY?***

**TOO MANY**
**IS BETTER THAN**
**TOO FEW.**

I will observe that if we're going to err, I would much prefer to err on the side of too many tags. When you don't have a tag for what your user wants to do, the user will "corrupt" the markup by using tags as formatting controls to get the desired formatting effect. That kind of corruption compromises your text base — not significant for documents with a short life expectancy, but very significant for documents with a long life.

We were a little slow in making available our tags for programming syntax markup (keywords and variables), and so our users used highlighted phrase tags instead. The formatting effect on the device in use at the time, the 3800 Model 1, was identical. The formatting on the devices in use today is radically different. Understand that we have thousands upon thousands of instances of programming syntax — and none of that markup would be valid today if we had let the use of highlighted phrase tags continue.

What intrigued me about this experience was how willingly, even eagerly, the publishing people in my own organization went into the then existing markup and corrected it. This was the first real evidence — observed many times since — that they were committed to ensuring the long-term validity of those source files. Of course, these are the same people who had done the text conversions I mentioned earlier, so they well understood the painful implications of short-term markup expediency.

Still another example of the "too few tags" syndrome was that our users originally were using the example (XMP) tag as an escape into unformatted text. We knew that in the future we would be setting examples in a monospace font in an otherwise proportionally-spaced document. We introduced the LINES tag as the general tag for unformatted text, and reserved the XMP tag for true examples of programming input and output.

In a similar vein, we had some initial resistance among writers to the use of the quote tag; again because we were using the 3800 Model 1, the net effect of the quote tag was the same as hitting the quote key on the keyboard. But keyboard quotation marks are another form of corruption of the markup when you consider devices that discriminate between open and close quotation marks.

The real question is "how many tags does an individual user have to deal with, and in what context?"

**Tag Counts in ISIL**

The last time I made an analysis, we had 206 tags in the ISIL vocabulary. But to understand our true status, a little more depth of analysis is required.





***TAG POPULATION IN ISIL***

| | | |
|---|---|---|
| **206** | TOTAL | |
| **181** | USERDOC VOCABULARY | |
| **43** | | ONE PER DOCUMENT |
| **138** | | REPEATING |
| **125** | GENERAL USERS (USERDOC SUBSET) | |
| **35** | BASIC ISIL | |
| **14** | | ONE PER DOCUMENT |
| **21** | | REPEATING |

There's a small number (about 6) of these tags that I consider to be gratuitous. Some of these were added to the language to get around a particular implementation restriction; the restriction has gone away, but the tags won't.

USERDOCs have a total vocabulary of 181 tags. Of those 181 tags, 43 are one-per-book tags (for example, the title page tag), if at all. That leaves us with 138 tags for the "body" of the book.

We are producing a "general user's guide," as compared to our "publications production guide," that documents a subset of the user document tags. This eliminates another 50-odd tags that are of interest only to publications people.

Our *Basic ISIL for Simple Documents* book, which we offer for new and occasional users, documents 35 tags, of which 14 are one-per-document (if at all). The tag set that represents the bulk of the markup for the casual user numbers 21. This 21 covers paragraphs, headings, general lists, highlighting, examples, and lines — adequate for the casual user, and a good starting point for the user who will eventually move up.

Still, returning to the 181 tags that can occur in USERDOCs, it's a lot. However, many of those tags exist in a very limited context. For example, our tags for messages and codes lists represent almost 10% of that total vocabulary. They are relevant only to someone who is doing such a list. As you are well aware, those lists can run to several hundred pages for a single product. When you are doing such a list, the tags take a short time (a matter of minutes) to "learn," but save an enormous amount of time in creating the list.

Another instance of this is our question-and-answer tags for independent study manuals. As tags go, these are relatively complex to learn and apply. But the number of users who actually do questions and answers is small and these users are specialists in the creation of such materials. In any case, the tags are not more complex than the problem dictates.

The motivations of your users, then, is a major factor in the "too many tags" debate.

While the potential certainly exists for there to be too many tags for even the proficient user to cope with effectively, I'm not concerned that we've reached that threshold, or even that we will, for this reason:

The vocabulary of "common" tags — tags used by general users — is no longer growing at a runaway rate; we already have much of what we need. Thus, the bulk of what we will be adding in the future are those, like the question-and-answer tags, intended for use by specialists.

Furthermore, our user documentation is designed so that a using organization can select from a "menu" those specialist tags that they want included in their local edition of the user's guide. Documentation subsets are an effective screen against perceived complexity.

### *How Many Attributes is Too Many Attributes?*

Another area in which usability can be profoundly affected is the number of attributes on a particular tag. I feel that, from a usability standpoint, this question is a far more serious consideration than the general question of how many tags you have; users are less comfortable with attributes than with tags. How many is too many is dependent on the nature of the tag. On the paragraph tag, one is too many (or at least one that a user has to consider very often).

***ATTRIBUTE POPULATION IN ISIL***

| | | |
|---|---|---|
| **206** | TAGS: | |
| | **107** | NO ATTRIBUTES |
| | **99** | ATTRIBUTES |
| **109** | UNIQUE ATTRIBUTES | |
| **278** | ATTRIBUTE OCCURRENCES | |
| | ID | OCCURS 34 TIMES |
| | REFID | OCCURS 27 TIMES |

...BUT ONLY 5 ATTRIBUTES IN "BASIC ISIL"

Of our 206 tags, 107 have no attributes at all. On the other hand, we have 109 unique attributes that can occur one or more times on the remaining 99 tags, for a total of 278 attribute occurrences in the language. Most tags that do have attributes have one or two.





We have six attributes on our figure tag, and if that is not already the limit for that tag, it is close enough. One of them, DEPTH, generates white space for artwork; we now have an ARTWORK tag with its own DEPTH attribute. Had we had that all along, we would not have needed DEPTH on the FIG tag, and would have only one way to specify depth for artwork, which is generally preferable. Likewise, the LABEL attribute on the figure tag, which allows for printing an art label in the middle of the space generated by the DEPTH attribute, would not be needed on the FIG tag.

That would get our attributes on the figure tag itself down to four (ID, PLACE, WIDTH, FRAME), which seems about right. They are all reasonable variables to deal with on a figure.

As it happens, the defaults on the FIG tag attributes are not optimum for our use; we inherit them from the starter set, and keep them that way for consistency with the starter set. Left to our own devices, we would almost certainly default the PLACE attribute to INLINE rather than float TOP. Novice users are always dismayed the first time they forget to specify the PLACE attribute and their figure takes off on its own.

At the far end of the spectrum, consider our document profile (DOCPROF) tag, which specifies numerous "style" options for the format of the output document. This tag has 19 attributes — too many by any standard — and it will almost certainly grow. However, this is a fill-in-the-blanks tag that occurs once per book, if at all; everything on it defaults to "normal" processing. We simply don't expect our users to "know" the tag; they only need to know that it exists, the kinds of things it controls, and how to call up the skeleton when they want it. Our general users, as compared to our publications people, don't even need to know it exists, and we don't tell them.

Our table tags are another instance where there is a lot to remember (both tags and attributes) for the new user, and the markup is somewhat finicky. One of our more enterprising users created a "table tag generator" with an XEDIT macro. This gives you a a fill-in-the-blanks screen and then goes off and generates the tags. Even our experienced users find this a time-saver.

The area in which we've experienced the longest learning curve is indexing. Not coincidentally, indexing is functionally quite rich, and that function is exercised through attributes.

***ATTRIBUTE OCCURRENCES***
***INDEXING TAGS***

| | I1 | I2 | I3 | IH1 | IH2 | IH3 | IREF |
|---|---|---|---|---|---|---|---|
| ID= | X | X | X | X | X | X | |
| REFID= | | X | X | | X | X | X |
| CIX= | X | | | X | | | |
| PG= | X | X | X | | | | X |
| PRINT= | | | | X | X | X | |
| SEEID= | | | | X | X | | X |
| SEE= | | | | X | X | | X |

Over and above the starter set, we've added one attribute (for cross indexing) on the primary entry tags, and we've extended the list of tags on which you can use the REFID attribute. This eliminated one seemingly arbitrary restriction that had been a source of confusion. (It wasn't really arbitrary. When we designed the starter set tags, we couldn't think of a valid use for it. Now we can.) The indexing tags vary between three and five attributes apiece, the attributes being selected from a total of seven. Our users have trouble keeping track of which attributes go with which tags.

Our users have problems conceptually with the uses of ID and REFID on the indexing tags; a single indexing tag can have both attributes, and this compounds the confusion. What I've observed with ID/REFID on the indexing tags is that users tend to overkill, using many more of these attributes than the application requires.

This is not to suggest that they can't or don't learn it — it just takes a little longer than most of the other areas.

While we're on the subject of indexing, I might mention that our cross-indexing attribute (CIX), which allows our users to associate subordinate entries with multiple primaries, is the one case so far in our language where we allow multiple occurrences of an attribute on a single tag. This is how it looks:

***MULTIPLE OCCURRENCES OF THE CIX ATTRIBUTE***

```
:IH1
CIX='starting a session'
CIX='session, starting a'
CIX='initializing a session'.
logon procedure
:I2.entering your userid
:I2.entering your password
```

```
initializing a session
    entering your password  5
    entering your userid  4
logon procedure
    entering your password  5
    entering your userid  4
session, starting a
    entering your password  5
    entering your userid  4
starting a session
    entering your password  5
    entering your userid  4
```





As it happens, the DCF control word (.GS EXATT) that executes attributes will execute as many occurrences of an attribute as you give it, so nothing special had to be done to implement this. In the case of CIX, the practical limit is the number of things you can put in the indexing buffer before it breaks.

Our message list and code list tags are two more with a large number of attributes, but they represent a special case, which I'll talk about in more detail later.

**The ID Attribute**

The ID attribute is in a category all by itself. The potential exists to put an ID attribute on every tag in a document; that is, every document element could have its own unique identifier, notwithstanding the fact that we only bother to process selected ones. We currently recognize the ID attribute on 34 tags.

For example, we use an ID attribute on the XMP tag for purposes of tracking examples to verify them; the ID is related to a particular test case. We can optionally print the ID with the example, but in any case we can use the cross-reference listing to locate the example in the document. This is just a trivial instance of what might be termed "utility functions" of GML that aren't necessarily manifest in the formatted document.

## *Generic Tags Versus Specialized Tags*

Another source of considerable debate is generic tags versus specialized tags — we have many instances of both types. For purposes of definition, a generic tag is one that is purely structural, with no implications about its content. For example, H1 (heading level 1) is a generic tag. A specialized tag is one that reflects the content. For example, in ISIL the PREFACE tag is a specialized tag that also has the properties of a level 1 heading.

***GENERIC AND SPECIALIZED TAGS***

```
:H1.PREFACE

:PREFACE.
```

```
:H1.WRITER'S LAMENT

:PREFACE.WRITER'S LAMENT
```

The preface in IBM manuals is supposed to contain information about the structure and usage of the book. As such, it is a valid entity that one might want to "operate on" in some application. In the two cases above the line, one could identify the preface from either instance — although as a general rule, we would want to identify an element from an analysis of the tag alone, and not from an analysis of the tag and its content.

However, we do not require that a preface actually be titled "preface" — text supplied with the tag overrides the generated text. (We have a number of tags, similar in nature to PREFACE, that have this property.) Thus, in the cases below the line, only the second would be identifiable as the preface. Any application that went looking for prefaces would fail to find the first one.

No one argues that all tags should be either generic or specialized, but we do examine the issue frequently when defining the language for an individual document element.

***WHEN TO GO "SPECIALIZED"***

- USER PRODUCTIVITY
- FLEXIBILITY
- OTHER USES
  - RETRIEVAL, SELECTIVE PROCESSING
  - FUTURE DEVICES
  - SEARCH ARGUMENTS
  - TRANSLATION

  ETC.

The decision points are basically these:

1. User productivity. Specialized tags (as opposed to generic tags with content) are faster to enter. Our messages and codes lists tags have about 10 specialized tags for the subheadings we use repeatedly in these lists — Explanation, User Response, System Action, etc. They represent an enormous saving in just plain keystrokes.

2. Flexibility. Specialized tags allow you to alter the formatting style for document elements more selectively. For example, we have a generic tag, QUALIF, for handling certain kinds of qualification within a document. We also have an IBMX tag, for IBM extensions to standard languages, that represents a specialized case of the QUALIF tag. While we currently handle both of them the same way, the distinction was made so that we could in the future handle them differently, without the need to analyze the tag content.

3. Other applications of the text. We attempt to anticipate other uses of our text and build into our language mechanisms that will facilitate those uses.





Among these other uses are future retrieval and selective processing applications. We can envision wanting to extract from a massive message list only those messages for which a "system programmer response" is indicated; the fact of a "system programmer response" tag enables that.

Some of our "future" applications are more mundane. For example, when ISIL was born, we supported only monospace devices that did not discriminate between open quotes and close quotes. But we knew that some day we would be dealing with devices that made that discrimination. So we invented the quote tag at the outset, telling people that it was an investment in the future.

Similarly, our "highlighted phrase 1" tag produces the same formatting effect as our "title citation" tag. But title citations are useful search arguments in a source file, when you want to verify that your title citations are current (which we have to do a lot).

Specialized tags are also a modest aid to translation; they cue the translator as to what's going on. Where they generate text, the translation need only be done once.

### *The End Tag Debate*

One topic that we work over from time to time is the whole subject of end tags. This is largely a theological debate, as our direction is well established and we aren't going to change. It goes like this:

***THE END TAG DEBATE***

GIVEN THAT WE HAVE MANY TAGS THAT REQUIRE AN END TAG, SHOULD WE NOT ALSO ALLOW ALL OTHER TAGS TO HAVE OPTIONAL END TAGS?

Our intuitive feeling is that we should, although we don't. The debate stems from such general questions as "what is the scope of a paragraph?" Clearly, another paragraph at the same level (that is, not a paragraph in a subordinate list) ends the current paragraph, as does a heading. But we treat any lists, examples, or inline figures as being part of (subordinate to) the preceding paragraph (that is, to the extent that we recognize the problem at all), and this is not necessarily the case. As a practical matter, in our current applications, it's a not a real consideration. But if you anticipate an application that operates on paragraphs as an entity, then the scope of a paragraph becomes a concern. A paragraph end tag would eliminate any doubts — although it could complicate matters for the material following the paragraph.

But even if we offered these optional end tags, few if any users would actually put them in, so you could not build a future application against the current text base on the assumption that they were there.

Incidentally, we consider our paragraph continuation tag to be unnecessary from a GML standpoint, but we have yet to alter the implementation so that it always works properly without it.

The end tag debate also reveals an anomaly about our heading tags. The heading tags do two things: they define a portion of the document at a structural level, ended by the occurrence of another heading at the

same or higher level; and they define the text of the heading at that level. If we were to introduce an end tag for the text of the heading (and we have applications where that would be useful), what are the implications of having "ended" that document element?

I said these debates were theological; I offer them only as examples of how we can spend so much time designing our language.

### *Consistency*

Consistency is as important as any other consideration in the design of a GML application. We attempt to be highly disciplined in examining this aspect of proposed new language; sometimes we've even sacrificed some minor elegance to the need to be consistent.

***SOME ISIL INCONSISTENCIES***

SOME WE DIDN'T FIX:

DEPTH ATTRIBUTE

FIGURE CAPTION AND TABLE CAPTION

ONE WE DID FIX:

CONDITIONAL AND UNCONDITIONAL MULTIPART FIGURES

We are guilty, however, of some inconsistencies. They've been around so long now, and there is such a heavy investment in the existing markup, that we'll probably live with them forever.

For example, we have two different meanings for the DEPTH attribute, depending on which tags it occurs on. This was a flat-out mistake.

As another example, captions on figures have their own FIGCAP tag, whereas the caption on a table is an attribute of the table tag. This came about as a result of our stealing our table tags from another internal implementation. We don't always look a stolen horse in the teeth.

Of course, it could just as readily be argued that the figure caption should also be treated as an attribute. I would be inclined to that view myself, except that users find tags less intimidating than attributes, and we already have six attributes on our figure tag. In any case, the figure tag dates back to release 1, when multiline markup was not supported; it simply wasn't feasible to specify the caption as an attribute at that time.

Both of these inconsistencies were introduced very early in the life of ISIL; they would not happen now, because we are much more disciplined in our design activity.





This is not to suggest that we've tolerated every inconsistency that has been introduced. We had a horrendous inconsistency in how we treated multipart figures, depending on whether they were unconditional (the figure is always to break at a specified point) or conditional (the figure is to break at a specified point if there is insufficient room on the page for the next segment). The two different approaches came about by way of historical accident; we didn't set out to make them different.

In any case, once the implementer advised us that he had figured out how to converge them into a single approach, we made that change. We continue to support the markup for the old way, but with warning messages that the markup is obsolete and should be changed. (In the early days of ISIL, we changed the language a number of times; wherever possible, we continued to support the old markup with a warning message. We found that most people actually did update the markup.)

If we had waited until we reached our current level of sophistication before going into production with ISIL, we would undoubtedly have eliminated most, if not all, of our inconsistencies. But if we had waited, we never would have reached our current level of sophistication. We learned by doing, and many of the things we did can't reasonably be undone. Just guessing, because I'd be hard-pressed to count even my own organization, there must be several million active pages marked up in ISIL today; that's an investment we don't toy with lightly.

## Some Specific Problems

I'm going to talk about a couple of the specific problems we've dealt with in the design, just by way of illustrating some of the problems you might encounter.

***SOME SPECIFIC PROBLEMS***

- MESSAGES AND CODES LISTS SUBHEADINGS
- QUESTION AND ANSWER TAGS
- THE DOCUMENT PROFILE TAG

### *Messages and Codes Lists Subheadings*

At first blush, you wouldn't think (or at least we didn't think) that messages and codes lists could be all that exciting. As it happens, I find myself calling upon them repeatedly to illustrate various points about ISIL. Anyway, they presented us with one of our knottiest design problems to date.

From the very early days of ISIL, we've had tags for the subheadings we use in these lists: XPL (explanation), URESP (user response), SYSACT (system action), and so forth. We started with about six of these, and the list has now grown to about ten, each an equally legitimate distinct entity. There may be one or two more still lurking in the woodwork; they will surface eventually and we will accept them.

That wasn't the problem.

***MESSAGE LIST SUBHEADING PROBLEM***

| | |
|---|---|
| **XPL** | Explanation<br>Cause<br>Reason |
| **URESP** | User Response<br>Recovery<br>What to Do |
| **SPRESP** | System Programmer Response<br>Administrator Action |

The problem came about when we got a request for a "CAUSE" tag and a "REASON" tag, because some books require different generated subheading text. It didn't take us very long to determine that, from a GML view, these were already covered by our XPL (explanation) tag. Similarly, people wanted to use "recovery" rather than "user response," and so on.

To control it at the level of an individual installation was a trivial matter — the local profile could be changed, and wouldn't even require an ISIL committee review. But it had to be controllable at at least the level of an individual book, and potentially at the level of an individual list.

We toyed for a long time with the idea of creating a generic "message list subheading" tag, where the content would determine the text. This was unacceptable for two reasons:

1. It involved too many keystrokes; users with reasonable, although different, requirements should not have to pay a productivity penalty.

2. It would be at the loss of the identity of this document element as the explanation of this message. We're all convinced that some day we will have an application that exploits this information.

We could not allow the residual text to override the generated text, as we do with the preface tag, again because of the keystrokes, and also because it would be incompatible with existing markup, as these tags are all implied paragraphs.





***MESSAGE LIST SUBHEADING SOLUTION***

```
:MSGL
XPL='REASON'
URESP='RECOVERY'
SPRESP='ADMINISTRATOR ACTION'.
.
.
.
:XPL.....
.
.
.
:URESP....
.
.
.
:SPRESP....
```

Our solution was to create, for each of these tags, a corresponding attribute on the message list tag itself, wherein the user could specify the override text. We make the user responsible for ensuring that the validity of the GML type is maintained.

This results, of course, in an enormous number of attributes on these list tags. But these attributes are all of a single class and parallel a known tag vocabulary, so the user has to learn only one concept to apply them all.

Incidentally, we've since come up with another application for this same type of attribute — again, overriding the text generated by subordinate tags.

### *Question and Answer Tags*

Another of our big debates came the day that the people who prepare tutorial and independent study materials requested question-and-answer "list" tags. The argument focused on the "listness" of the entity. In practice, there were a number of different ways of treating the numbering of these things, and the proposed language was attempting to accommodate all of these variations.

***QUESTION AND ANSWER PROBLEM***

WHEN IS AN ENUMERATED LIST NOT A LIST?

WHEN THE ENUMERATION OF ITEMS IS INCIDENTAL.

We finally concluded that the fact that these items were numbered in the output document was totally incidental — just as the numbering of headings does not make of them a list. It's merely a retrieval device. Our ultimate solution here was to decide that the markup would not treat them as lists, and that the implementation would handle two "standard" numbering approaches (based on the page numbering scheme used in the book, of which we support two). Local installations could implement local variations, if need be.

***THE LESSON OF QUESTIONS AND ANSWERS***

IT IS NECESSARY TO

- MEET USER REQUIREMENTS
- PRODUCE A REASONABLE OUTPUT RESULT

IT IS NOT NECESSARY TO

- SUPPORT EVERYTHING THAT ANYONE MIGHT EVER HAVE DONE IN THE PAST

The lesson here is one we've had to relearn a couple of times:

> It is necessary to meet rational user requirements, and it is necessary to produce an output result that is generally accepted as reasonable. It is *not* necessary to support every way that anyone might ever have done things in the past (especially when there haven't been a lot of constraints on how things were done in the past). That's a trap.

We have no rules against local modifications to the formatting; our rules deal with the integrity of the source file.





### *The Document Profile Tag*

Over the life of ISIL, we had externalized a number of "style" variables, either as runtime options, which are error-prone, or by documenting internal symbols and telling people how to reset them to new values. On the table were many more, and we could see ourselves getting locked into carrying these symbol names in perpetuity. The symbols lacked a certain user-friendliness as well, as they were full of @ characters and # characters, which are not particularly mnemonic for most of us.

Now style is not really something that you want to vary every time you format a document; it's generally fixed for at least some period in the life of the document. We needed — given the range of uses we support — some reasonable way to allow user access to these legitimate style variables. Our solution was the document profile tag, DOCPROF. This is the tag with far too many attributes that I mentioned earlier.

Here's what's currently on the DOCPROF tag; we fully expect it to grow:

***THE DOCUMENT PROFILE TAG***

```
:DOCPROF  BODYHD1=
          CAPLOC=
          DIALOG=
          FBC=
          HD1PREF=
          HEADNUM=
          HYPHEN=
          JUSTIFY=
          LAYOUT=
          LDRDOTS=
          MCINDENT=
          MCSPACE=
          PTOC=
          PUNCT=
          RHRFRULE=
          STYLE=
          TIPAGE=
          TOC=
          XREFPAGE=
```

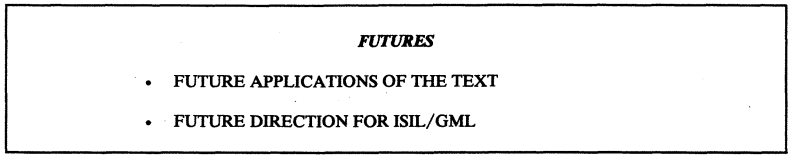
| **This attribute...** | **establishes...** |
|---|---|
| BODYHD1= | what is generated for H1s in the body of the document. |
| CAPLOC= | placement of figure and table captions. |
| DIALOG= | formatting style of user-system dialogs. |
| FBC= | serial page numbering or by chapter. |
| HD1PREF= | style of cross references to H1s. |
| HEADNUM= | automatic numbering for headings. |

| | |
|---|---|
| HYPHEN= | hyphenation. |
| JUSTIFY= | justification. |
| LAYOUT= | the basic column layout of the document. |
| LDRDOTS= | leader dots in the table of contents and figure list. |
| MCINDENT= | indention for text in messages and codes lists. |
| MCSPACE= | spacing between messages/codes in messages and codes lists. |
| PTOC= | levels of headings for partial tables of contents. |
| PUNCT= | punctuation placement with respect to closing quotation marks. |
| RHRFRULE= | rules on running headings and footings. |
| STYLE= | named document style. |
| TIPAGE= | the basic layout of the title page. |
| TOC= | levels of headings for table of contents. |
| XREFPAGE= | use of page numbers in cross references. |

The DOCPROF tag falls into a category known as "coexistent structures." That is, rather than describing the structure of the source document, it describes the structure of the output document.

## Futures

There are two kinds of future considerations for ISIL/GML — future applications of the information base, and future directions for ISIL itself.




I'm not going to spend much time on the subject of future applications of the text "data base." By "data base" here I mean a set (a very large set) of consistently marked-up source files. You can speculate as well as I can as to the uses we might make of it.

Information retrieval applications come to mind immediately — something that simply wasn't feasible before because of the infinite variety of the source documents. Another potential application is the distribution of publications in machine-readable form. You must understand, however, that graphics is a rapidly growing element in our information "data base," and the ISIL solution applies only to text. Both of these applications require that the graphics problems find some solutions — including, but not limited to, the widespread availability in the field of devices that can adequately display or print the graphic content.

The future of ISIL/GML itself is also potentially rich. For instance, color separation is a major item on our current wish list, and some of our language definition reflects our expectation that some day we will get it. Of course, automated color separation requires some new support in the underlying processor. But even without this support, we can still produce a "color guide" version of the document that shows where the





color separation should occur in the final version of the document, making it easier to prepare the final version manually.

One of my futuristic fantasies has to do with programming syntax presentation. Where today we hard-code brackets and braces (or even worse, paste them on), I hope one day to have a rigorous GML markup for programming syntax. Then a sophisticated implementation could format it with the traditional brackets and braces or in some of the alternative presentation styles we play with from time to time.

If you then have the language designers do their original language syntax specification using the GML, you eliminate the potential for introducing error or ambiguity when you translate the language designer's specification format to the presentation style you're using in the user's guide.

Further, I'd like markup that relates the elements of the syntax to the discussion of them. One potential here is to create help panels in a hierarchical presentation, derived from linear material.

The challenge here is not particularly in designing the GML itself. We have a pretty good idea of what it would have to look like. But the "sophisticated implementation" is beyond our current capabilities.

A rigorous GML markup for programming syntax also enables other processing applications, such as evaluating programming language complexity from an analysis of the syntax.

I'm sure that other users of ISIL who don't do zillions of programming syntax presentations, but do something else, would come up with equally exotic futures.