

# **InfoML Specification, Version 0.83**

**Gregg Williams**

---

# **InfoML Specification, Version 0.83**

Gregg Williams

---

---

---

# Table of Contents

1. Overview .....	1
1.1. Introduction .....	1
1.2. Goals .....	1
1.3. Features .....	1
1.4. An Example Infocard .....	2
2. Understanding the InfoML Specification .....	4
2.1. Introduction .....	4
2.2. The Two Levels of the InfoML Specification .....	4
2.3. The Syntax of InfoML (Level 1) .....	4
2.4. The Semantics of InfoML (Level 2) .....	5
2.4.1. The name Attribute .....	5
2.4.2. Reserved Values for the name Attribute .....	6
2.4.3. Developer-Specific Top-Level Subelements .....	6
2.4.4. Infocard Content Data .....	6
2.4.5. Infocard Structure Data .....	7
2.5. The InfoML Top-level Subelements .....	7
2.5.1. The <cid> Element .....	7
2.5.2. The <pid> Element .....	8
2.5.3. The <selector> Element .....	8
2.5.4. The <tag> Element .....	8
2.5.5. The <body> Element .....	8
2.5.6. The <context> Element .....	9
2.5.7. The <comment-on> Element .....	11
2.5.8. The <pointers> Element .....	11
2.5.9. The <special> Element .....	11
2.6. Another Example Infocard .....	12
3. Customizing Infocards .....	15
3.1. Levels of Customization .....	15
3.2. Application Handling of Customized Infocards .....	15
4. Design and Implementation Issues .....	16
4.1. Adding InfoML to your Application .....	16
4.2. Minimizing the Use of Private Data .....	16
4.3. Standard Locations for Information .....	17
4.3.1. Standard Locations for “Visible” Information .....	17
4.3.2. Standard Locations for “Invisible” Information .....	17
A. The InfoML Resource Indicator (IRI) String .....	18
A.1. Some Examples .....	18
A.2. Terminology, Rules, and Usage .....	18
B. The InfoML 0.83 Schema .....	20
C. Rules for the InfoML 0.83 Semantics (Level 2) .....	27
D. Important Elements and Attributes for InfoML 0.83 .....	29

---

## List of Figures

1. A simple <infoml> element .....	2
2.1. A more complex infocard. ....	12

---

## List of Tables

1.1. Features and benefits of InfoML. ....	1
2.1. Interpreting the contents of the <code>special</code> element. ....	12
C.1. Regular Rules for Level 2 of the InfoML Specification ....	27
D.1. Reserved values for InfoML 0.83 ....	29

---

# Chapter 1. Overview

## 1.1. Introduction

Remember those 3x5" notecards you used in college when you were researching a term paper? You would jot down some notes or a quotation on a notecard, then add a title on the top line. Once you'd done enough research, you would start arranging your cards in piles based on their content. Or maybe you would tape them to a wall and move them around until you had them arranged in a way that made sense. It wasn't a perfect solution—especially if you felt that one card belonged in several places—but it worked.

An InfoML file is like a set of notecards, in the same way that a web site is like a newspaper—the metaphor is loosely descriptive, but it doesn't tell the whole story. So, just as a web site can offer so much more than a newspaper can, an InfoML file can be far more than the electronic equivalent of a stack of notecards. Informally, the unit of information that is stored in one “electronic notecard” is known as an *infocard*.

The InfoML format is very versatile, and people can create custom infocard formats to meet specialized needs. However, the InfoML Standard describes a standard infocard format that will meet most people's needs; any end-user software that is *InfoML-friendly* (that is, that supports the InfoML Standard) will be able to recognize and manipulate any infocard that uses the standard format. Because of the extensibility of the InfoML format, end users can attach "add-on" content to any standard infocard (think of adding "sticky notes" to a regular notecard) without destroying the ability of InfoML-friendly software to recognize and manipulate it correctly. And InfoML becomes even more interesting when people start using it to share information.

## 1.2. Goals

The project to create the InfoML Standard has several goals:

- First, to create a system for encapsulating units of knowledge in a way that facilitates organizing, searching, and modifying individual infocards, as well as providing a mechanism for combining multiple infocards in meaningful ways.
- Second, to create a system that is easily understandable for simple uses but is expressive enough to handle more complicated situations.
- Third, to create a system that maximizes the usefulness of the individual infocard to both its creator and to anyone else who gets a copy of it.

## 1.3. Features

Consistent with the goals expressed in the previous section, the current implementation of InfoML includes, among others, the following features:

**Table 1.1. Features and benefits of InfoML.**

Feature	Benefits
Information is stored in an XML format.	Doing so increases the longevity and usefulness of the data, enables developers to use the many existing XML-based tools,

Feature	Benefits
	and increases the likelihood of InfoML becoming widely used.
Information is captured in elements that conform to a schema.	The presence of a schema makes it easier for developers to write useful software and to reuse the captured information for new purposes.
Every infocard contains a globally-unique identifier.	This enables developers and users to build up structures of information (by creating infocards that point to other infocards).
Infocards are self-contained—that is, isolated from a larger context, they contain or point to all the information needed to fully understand the infocard’s content.	This makes infocards easier to understand and more likely to be shared with others.
Infocards are independent—that is, changing an infocard or creating a new one should not force any other existing infocard to be modified.	Without this feature, the ability to give a single infocard or an arbitrary set of infocards to another person would be impaired.
Each infocard can be categorized in a variety of ways through the addition of any number of informally-chosen keywords.	The unrestricted use of keywords is a simple form of categorizing that users will intuitively understand.
The InfoML Standard defines a standard infocard structure that all InfoML-friendly applications can process correctly.	This encourages the creation of many diverse applications, all of which can exchange data—thus making InfoML-based information more useful.
Users can annotate infocards with additional data in a way that does not break existing InfoML-based software.	This makes infocards more useful to individual users, who can “attach” whatever additional information they need to (think of notecards that can have any number of “sticky notes” attached to them). Properly designed software can manipulate infocards without losing the user’s “attachments.”
The InfoML Standard is designed to allow evolution of the standard, while maintaining backward compatibility with existing InfoML-based data.	This means that InfoML can evolve to meet customer needs without causing any backward-compatibility problems.
Infocards are used primarily to store formatted or unformatted text, but they can “include” any data or media that can be pointed to using a URL.	This feature increases the range of situations to which InfoML may be applied.
Users may optionally include authorship information in a infocard, and they may create modified infocards in a way that maintains a link to the original infocard.	These features enable infocards to be fluid over time, while still documenting an idea’s evolution and authorship.

## 1.4. An Example Infocard

Before proceeding further, it would probably help to see an example of an `<infoml>` element. Even without knowing InfoML, you can easily spot the infocard’s card ID (`cid`), several keywords (marked by the string “`key`”), the body of the infocard (marked by the string “`content`”), as well as information about the source of the body and the date that the infocard was created.

**Figure 1. A simple `<infoml>` element**

```
<infoml encoding="UTF-8" version="0.83">
  <cid>abbyw.aol.com_101</cid>
  <selector name="cardtype">idea</selector>
  <selector name="key">love</selector>
  <selector name="key">age</selector>
  <selector name="key">inner beauty</selector>
  <selector name="key">woman</selector>
  <body name="content">
    <p>Then he bends to embrace her [an old woman]...and when he
      looked at her, in the whole world was not a young woman of
      gait more graceful.</p>
  </body>
  <context name="source">
    <agent role="author">
      <first>Standish</first>
      <middle>H.</middle>
      <last>Freeman</last>
    </agent>
    <container>
      <name-part>Silva Gadeilica</name-part>
    </container>
    <location>
      <id-part unit="volume">II</id-part>
      <point unit="page">114</point>
    </location>
  </context>
  <context name="this-card">
    <date-created>2004-10-10</date-created>
  </context>
</infoml>
```

---

# Chapter 2. Understanding the InfoML Specification

## 2.1. Introduction

The InfoML Specification is more than just a definition of what constitutes a legal `<infoml>` element. (Remember, the informal name for the information contained in one `<infoml>` element is an *infocard*.)

To give InfoML the desired extensibility, it was necessary to add to the InfoML schema (which specifies whether or not a given `<infoml>` element is syntactically valid) an additional set of rules that add the desired semantics to the InfoML Specification. The following sections discuss this in further detail.

## 2.2. The Two Levels of the InfoML Specification

Why does the InfoML Specification have a syntactical level (Level 1) and a semantic level (Level 2)? There are two reasons for this. The first is that the desired structure of valid, useful infocards was dictated by the desire for several levels of extensibility.

The second reason has to do with the choice of schema for expressing this structure. There are several schema notations for describing the structure of XML. The most commonly used schema notation is XML Schema, and for this reason, the majority of development tools support XML Schema. Unfortunately, XML Schema cannot express all the dependencies and relationships that InfoML needs—hence, the need for the second, semantic level of the InfoML Specification.

Existing XML validation software can confirm that a given piece of XML code is, according to the InfoML schema document, valid (denoting Level 1 compliance). It will be necessary, however, to create new software to confirm that a piece of syntactically-correct InfoML is semantically valid (denoting Level 2 compliance).

## 2.3. The Syntax of InfoML (Level 1)

The syntax of a valid `<infoml>` element is defined by a document called an XML schema. The schema for InfoML version 0.83 is listed in Appendix B, “The InfoML 0.83 Schema,” but it can be summarized as follows:

1. Exactly one instance of the `<cid>` (card ID) element, followed by
2. zero or more instances of the `<pid>` (private ID) element, followed by
3. zero or more instances of the `<selector>` element (for keywords and the infocard’s card type), followed by
4. zero or more instances of the `<tag>` element (for the infocard’s title), followed by
5. zero or more instances of the `<body>` element (with infocard’s main content), followed by
6. zero or more instances of the `<context>` element (for information on the main content’s source), followed by
7. zero or more instances of the `<comment-on>` element (if this infocard was created to comment on the information in one or more other infocards), followed by

8. zero or more instances of the `<pointers>` element (used only when establishing relationships to other infocards), followed by
9. zero or more instances of the `<special>` element (used for information optionally entered by the user).

Given this syntax, the following is a valid (though useless) infocard:

```
<infoml version="0.83" encoding="UTF-8">
  <cid>greggw.infoml.org_776</cid>
  <body />
</infoml>
```

(Assume that `greggw.infoml.org_776` is a valid value for the `<cid>` element.)

Similarly, a random combination of the above elements would also be a valid, though equally useless, infocard.

### Note

Although the `<infoml>` element is the cornerstone of the InfoML Standard, the InfoML schema includes a “wrapper” element, `<infoml-file>`, whose only purpose is to contain 1 or more `<infoml>` elements. In this way, a valid XML file with an `<infoml-file>` element as its roots can contain any number of infocards.

### Note

In some situations, users may modify an infocard enough to warrant the creation of a new infocard. Users who want to keep track of this history of the modified infocard can use the `based-on` attribute of the `<infoml>` element for this purpose. This element should be used to contain the card ID (`cid`) value of the infocard on which the current infocard is based.

Useful infocards are made by imposing a set of rules on the total set of possible infocards as defined by the InfoML schema. These rules comprise Level 2—that is, the *semantics*—of the InfoML Specification.

## 2.4. The Semantics of InfoML (Level 2)

The rules describing the semantics of the InfoML Standard are given in Appendix C, “Rules for the InfoML 0.83 Semantics (Level 2).” The sections that follow describe the main ideas of Level 2 of the InfoML Specification. Once you understand these ideas, you will be able to make use of the rules set forth in this appendix.

### 2.4.1. The name Attribute

If you look at the sample infocard in Figure 1, “A simple `<infoml>` element,” you will notice that the two `<selector>` elements in lines 3 and 4 differ in the value of each element’s `name` attribute. In line 3, `name` has the value “`cardtype`”; because of this, the element’s value, the string “`idea`”, denotes that the current infocard is an idea infocard. (There are six kinds of infocards: idea, fact, opinion, definition, narrative, and generic.) In contrast, in line 4, `name` has the value “`key`”, meaning that this element’s value, “`love`”, is one of this infocard’s keywords.

What is happening here? In essence, the use of the `name` attribute enables the `<selector>` element to serve multiple functions: Its value (contents) is interpreted based on the value of its `name` attribute.

### Note

To promote brevity and readability, this document will use  $x/y$  to be interpreted as “the `<x>` element that has a name attribute with the value `y`” (i.e., `<x name="y"> . . . </x>`).

## 2.4.2. Reserved Values for the name Attribute

The pattern of having elements with a name attribute is one element of InfoML’s extensibility. Level 2 of the InfoML Standard specifies, for certain elements, a small number of *reserved values* for that element’s nameattribute—for example, in the case of the `<selector>` element, the values `cardtype` and `key`. See Appendix C for a complete specification of each element’s reserved values.

## 2.4.3. Developer-Specific Top-Level Subelements

If developers of InfoML-friendly software want to add custom data to infocards, they can do so in an appropriate top-level subelement by setting that subelement’s name attribute to an InfoML Resource Indicator (IRI) string. (See Appendix A for a description of IRI strings.) Any subelement used in this way is said to be *developer-specific*.

To give an example, suppose that developers at Infocorp are using the InfoML format to store information about every part in their inventory. To be able to do this, they create the IRI string “`inventory.infocorp.com_partno`” and decide to store the part’s ID string in a developer-specific `<pid>` (private ID) element. Each infocard will then have a `<pid>` element that looks like this:

```
<pid name="inventory.infocorp.com_partno">(part  
number goes here)</pid>
```

If the Infocorp company wishes to do so, it can publicize the format and usage of its developer-specific `<pid>` element, so that other companies can create software and data that understand this information. Of course, special software must be written to process such elements; a generic InfoML-friendly application will skip over such elements (though they should never delete them).

## 2.4.4. Infocard Content Data

As stated earlier, there are six kinds of infocards: idea, fact, opinion, definition, narrative, and generic. Except for some minor exceptions (covered elsewhere in this document), these different kinds of infocards are structurally identical. Two kinds of data can be stored within an `<infoml>` element: *content* and *structure*.

The following InfoML top-level subelements contribute to the content of an infocard:

- `<cid>`
- `<pid>`
- `<selector>`
- `<tag>`
- `<body>`
- `<context>`
- `<special>`

The value of the `selector//cardtype` element determines what kind of infocard you are looking

at. For example,

```
<selector name="cardtype">opinion</selector>
```

means that the current infocard is an opinion infocard.

Any InfoML-friendly application will treat these cards the same, so assigning one content-name or the other to a given infocard is a decision made by the infocard's creator (usually based on the infocard's content and the creator's wishes). Having a choice of values for each infocard's `selector/ /cardtype` element makes it possible for any user, at some future time, to select a group of infocards based on their `cardtype`.

## 2.4.5. Infocard Structure Data

One driving force behind InfoML is the belief that computers should be more useful in helping people process ideas. InfoML would not be fully reaching for this goal if it stopped at the level of individual ideas. Much of the task of processing ideas involves specifying relationships between different ideas and building collections of selected individual ideas into larger, meaningful groupings of ideas. For this reason, the InfoML Standard includes *structure data*—that is, data within an infocard that connects it to other infocards. The following InfoML top-level subelements contribute to the structure content of an infocard:

- `<comment-on>`
- `<pointers>`

The next two sections describe the two structure elements that may optionally be contained within an infocard.

### 2.4.5.1. The `<comment-on>` Structure Element

It is useful to note that the content of one infocard may sometimes be *based on* or *derived from* the content of one or more other infocards. This relationship is expressed by a `<comment-on>` element, which points to the infocard(s) to which the current infocard is a comment. These pointers are made by using the `cid` value of the parent card or cards.

### 2.4.5.2. The `<pointers>` Structure Element

The `<pointers>` element is meant to be used to build hierarchies of infocards; it can also be used to create other organized data structures where each node of the data structure is an infocard. For more details, see section 2.5.8, “The `<pointers>` Element.”

## 2.5. The InfoML Top-level Subelements

This section includes brief descriptions of each of the top-level subelements and notes on how they should be used.

### 2.5.1. The `<cid>` Element

The purpose of the `<cid>` (card ID) element is to provide a unique string that can be used anywhere to identify the infocard associated with it. The string used for this purpose is an InfoML Resource Indicator (IRI) string that is globally unique—that is, this string must not be used as the value of any other infocard in the world. In most cases, generating the appropriate IRI string is the responsibility of the software that is being used to create the infocard. For more information on IRI strings, see Appendix A.

The `<cid>` element is one of the few elements that is required to be in a valid infocard (that is, an infocard that meets both Level 1 and Level 2 of the InfoML Specification). Because it does not have a name attribute, developers cannot create developer-specific versions of the `<cid>` element; however, the `<pid>` element exists for that purpose.

## 2.5.2. The `<pid>` Element

The purpose of the `<pid>` element is to enable developers to create their own “private” identifiers for the infocards that their software creates; however, developers are free to use this element however they wish. An infocard may contain multiple `<pid>` elements, as long as each has a different value in its name attribute.

As is the case with all developer-specific elements, the developer should set the value of the element’s name attribute to an appropriate value. For details, see section 2.4.3, “Developer-Specific Top-Level Subelements.”

## 2.5.3. The `<selector>` Element

The purpose of any `<selector>` element is to facilitate the selection of one subgroup of infocards from all remaining infocards.

The InfoML Standard specifies “key” and “cardtype” as reserved values for this element’s name attribute. With the “key” value, the `<selector>` element contains a single keyword associated with that infocard; an infocard can include any number of keywords.

When the name attribute is “cardtype”, the value of the `<selector>` element for a given standard infocard must be one of the following values: “idea”, “fact”, “opinion”, “definition”, “narrative”, or “generic”. Custom infocards have a different value for the name attribute; see section 3.1, “Levels of Customization,” for details.

## 2.5.4. The `<tag>` Element

The purpose of any `<tag>` element is to provide one or more forms of brief human-readable content—usually a summary or some meaningful subset of the body content—for a given infocard. The InfoML Standard specifies “title” and “descriptor” as reserved values for this element’s attribute.

If you think of an infocard as an electronic notecard, then the `tag//title` and `<body>` elements constitute the title and contents of the notecard, respectively. In addition, developers can create their own developer-specific tags if they need to do so.

## 2.5.5. The `<body>` Element

The purpose of any `<body>` element is to provide the infocard’s main content. This element contains the “unit of information” that is the purpose of an infocard. In a simple infocard, all other content is metadata that enriches the meaning or value of the content in the `<body>` element. This is a mixed element that supports the following text styles: `<em>` (emphasis, usually rendered as italic), `<strong>` (bold), `<code>` (fixed-width font), `<a>` (hyperlink), and `<pre>` (preformatted).

The InfoML Standard specifies “source” and “notes” as reserved values for this element’s name attribute. The `body//source` element is the true contents of the infocard—that is, it should hold the “unit of information” mentioned earlier. The `body//notes` element is optional. In an InfoML-friendly application, a “Notes” field should always be made available to the user for every infocard; this provides a built-in way for all InfoML-friendly applications to provide users with a simple way to add secondary content to their infocards.

The `exact` attribute of this element is used to indicate whether or not the content in the `<body>` ele-

ment is an exact representation of the actual content pointed to by the `<context>` element. This attribute can have the values “true” and “false”; it defaults to “true”. You might, for example, want to set this attribute to “false” when the content of the current infocard is a paraphrase or summary of the actual content.

## 2.5.6. The `<context>` Element

The purpose of any `<context>` element is to provide information about the content in a corresponding `<body>` element—in particular, where the content comes from and what person or people are associated with it. In fact, a `<context>` element should exist in an infocard *only if* it has a corresponding `<body>` element; this is implemented by having the name attribute of the `context` element have the same string value as the name attribute of the `<body>` element.

Earlier versions of InfoML were concerned exclusively with capturing quotations from books, and earlier versions of the `<context>` element simply contained book, author, and page-number information. However, as the scope of InfoML grew to include other sources of information (e.g., journal articles, movies, web sites, and so on) and higher levels of detail, this element became more complex, and the names of its elements became more general. In addition to `<date-created>` and `<date-modified>` elements, the `<context>` element may contain three important subelements: `<agent>`, `<container>`, and `location`.

### 2.5.6.1. The `<container>` Element

The `<container>` element specifies where the infocard’s main content (that is, the content of the `body//source` element) comes from. (Actually, this is a simplified explanation. The container being referred to varies with the value of the name attribute of the containing `<context>` element; see section 2.5.6.4, “The `context//source`, `context//middle`, and `context//original` Elements,” for details.)

This element has an optional `type` attribute, which describes the type of container—for example, “book”, “article”, “lecture”, “movie”, and so on. This description is for the user’s benefit and can be any text string.

### 2.5.6.2. The `<agent>` Element

`<agent>` elements are used to identify the entities associated with a given container. For books, these could be the container’s author(s), editor(s), or other people associated with the book. Other types of containers will have other kinds of agents—for example, a movie might have associated with it a director, several producers, and the name of the company that is distributing the movie.

The kind of relationship between the agent and the container is described in a mandatory `role` attribute. This description is for the user’s benefit and can be any text string.

### 2.5.6.3. The `<location>` Element

The `<location>` element indicates a specific location within the container. This location can be a combination of *points* and *ranges*, each of which has a given *unit*. For example, if the container is a book, a `location` element may point to pages 3, 6, and 8-10 (where “page” is the unit). If the container is a film, the units might be “hour:minute:second”. This description is for the user’s benefit and can be any text string.

### 2.5.6.4. The `context//source`, `context//middle`, and `context//original` Elements

The most basic `<context>` element is the `context//source` element, and it is the only `context` element that most infocards will need. Its purpose is to provide metadata about the content in the `body//source` element (note that both elements have the same value for their name attribute—namely,

“source”).

In informal situations, the `context//source` element is sufficient. However, when there is a need for more rigorous documentation of the infocard’s main content, the `context//source` element is not enough. Consider the following hypothetical situation. Book A (written by authors A1, A2, and A3) contains the text “In book C, author C1 says, ‘XYZ is true.’” The footnote to this text indicates that that the text was found in book B (written by author B1).

If you need to document the content “XYZ is true” completely, you need to document three different containers: books A, B, and C. The InfoML Standard allows you to do this by providing three reserved values for the name attribute of the `<context>` element:

- “source”, which refers to the container in which the content was actually found (in this case, book A)
- “original”, which refers to the container that first contained the content (in this case, book C), and
- “middle”, which refers to any intervening containers that you need to include to fully document the content (in this case, book B).

With one exception (explained in section 2.5.6.7, “Vertically Stacked `<context>` Elements”), a given infocard should contain only one `context//source` element and one `context//original` element. If needed, the “chain” between these two elements can be filled by one or more `context//middle` elements. Only an extremely convoluted situation would necessitate the use of a second `context//middle` element.

### 2.5.6.5. The “speaker” Reserved Value

One more situation deserves mention. Assume you have the hypothetical situation described earlier, but you’re documenting the content “XYZ is true” less completely. If your infocard contains only a `context//source` element (which refers to book A, written by authors A1, A2, and A3), how do you document that the content was said by a different person, C1?

The solution is to have a reserved value, “speaker”, for the `role` attribute of the `<agent>` element. In this example, then, the solution is for the `context//source` element to contain four `agent` elements, one for each of the authors (using `<role>=“author”` in each case) and one (using `role=“speaker”`) for person C1.

### 2.5.6.6. The `context//this-card` Element

The optional `context//this-card` element has a special purpose in an infocard. When it is present (only one such element is allowed), it is used to store information about the infocard itself. Such information (all of which is optional) includes

- the date of the infocard’s creation,
- one or more instances giving one date when the infocard was modified, and
- the name, address, and electronic contact information of the person who created the infocard (stored as an `<agent>` element, where its `role` attribute has the value “creator”).

### 2.5.6.7. Vertically Stacked `<context>` Elements

The need for rigorous documentation of an infocard’s content causes one more rule to be added to the set of rules regarding `<context>` elements. Consider the following hypothetical situation. In book A (written by authors A1, A2, and A3), each chapter contains an academic paper, each of which has a dif-

ferent set of authors. If you want to fully document content from, say, paper B (written by authors B1 and B2), you need to document two contexts, book A and paper B. You can think of book A and paper B as being “vertically stacked,” with book A being “on top of” paper B.

The InfoML Standard handles the situation for `context//source` and `context//original` elements (but not `context//middle` elements). The solution is to allow multiple elements of the same type in direct succession within the infocard, with the understanding that the first such element documents the “top” (or enclosing) container and the last such element documents the “bottom” (or innermost enclosed) container. In the hypothetical situation described earlier, the result would be an infocard containing two “vertically stacked” `context//source` elements, the second immediately following the first. The first `context//source` element would document book A, while the second would document paper B.

## 2.5.7. The `<comment-on>` Element

The purpose of any `<comment-on>` element is to store information that indicates that the current infocard is *based on* or *derived from* one or more other infocards. If a given InfoML-friendly application allows users to create infocards commenting on other infocards, it should also provide an easy way to access all the infocards that are comment infocards for a given infocard.

The InfoML Standard specifies the string “source” (paralleling the `context//source` element) as a reserved value for this element’s name attribute. Except for the absence of the name and quality attributes, the `<comment-on>` element is identical to the `pointers` element; see the next section for details.

## 2.5.8. The `<pointers>` Element

The purpose of any `<pointers>` element is to facilitate the creation of meaningful structures of infocards. A `pointers` element contains an ordered list of nodes; each node contains a pointer to another infocard (specifically, the value of that infocard’s `<cid>` element), as well as an optional content element that can contain the same content as a `<body>` element. The intent is that the `<pointers>` element makes it possible for the creator of the element to annotate each pointer-to-another-infocard with some useful information about the infocard being referenced.

Three attributes of the `<pointers>` element deserve mention. The required name attribute must contain one of the following values: “sequence”, “unordered-list”, “ascending-list”, or “descending-list”. The value used specifies the nature of the relationship among the nodes in the `<pointers>` element. If the name attribute value is “ascending-list” or “descending-list”, an optional quality attribute describes the quality that is ascending or descending, respectively. For example, if the value of the name attribute is “descending-list” and the value of the quality attribute is “height”, the reader of this infocard can infer that the infocards being pointed to are listed (in some way that makes sense) in order of decreasing height. Also, the optional notes attribute provides a place to store information about the entire element—for example, a reminder of how the infocards being pointed to relate to each other.

## 2.5.9. The `<special>` Element

The purpose of the optional `<special>` element is to provide another opportunity for users to add their own custom data to a given infocard. Unlike the `body//notes` element, the data in a `special` element can be interpreted as one of the following data structures: list, two-dimensional array; three-dimensional array; property list (one value per key); or irregular property list (one or more values per key).

The InfoML Standard specifies “source” as a reserved value for this element’s name attribute. Every InfoML-friendly application should be able to display the data in the `special//source` element, though the method for doing so is left out to the application’s implementer.

The `<special>` element contains a single string. However, this string is interpreted according to the

values of four different attributes:

- the `table-type` attribute, which contains one of the following reserved values: “list”, “2d-array”, “3d-array”, “prop-list”, or “irreg-prop-list”, and
- the `major-delimiter`, `minor-delimiter`, and `tertiary-delimiter` attributes, each of which contains a string (not necessarily a single character) used to delimit subfields within the contents of the `<special>` element.

The table below shows how the value of the element should be interpreted.

**Table 2.1. Interpreting the contents of the `special` element.**

table-type value	value of special element
list	v1###v2###v3
2d-array	r1c1@r1c2@r1c3###r2c1@r2c2@r2c3 (a 2-by-3 array)
3d-array	r1c1v1~r1c1v2@r1c2v1~r1c2v2@r1c3v1~r1c3v2###r2c1v1~r2c1v2@r2c2v1~r2c2v2@r2c3v1~r2c3v2 (a 2-by-3-by-2 array)
prop-list	k1@v1###k2@v2###k3@v3 (three key/value pairs)
irreg-prop-list	k1@v11~v12###k2@v21###k3@v31~v32~v33 (the three keys have two, one, and three values, respectively)
	Assume <code>major-delimiter</code> =“###”, <code>minor-delimiter</code> =“@”, <code>tertiary-delimiter</code> =“~”, v=value, r=row, c=column, k=key.

## 2.6. Another Example Infocard

Now that the InfoML Standard has been explained, you might want to look at a more complex (but entirely fictional) `<infoml>` element:

**Figure 2.1. A more complex infocard.**

```
<infoml version="0.83" encoding="UTF-8">
  <cid>greggw.infoml.org_776</cid>
  <selector name="cardtype">idea</selector>
  <selector name="key">negative therapy</selector>
  <selector name="key">externalizing</selector>
  <selector name="key">problems</selector>
  <tag name="title">Externalizing the problem</tag>
  <body name="source" exact="true">
    <p>"The problem is the problem, the person is not the problem" is
      the relevant adage here. By separating the person from the
      problem, the therapist is able to motivate clients into new
      action.</p>
  </body>
  <body name="notes">
    <p>Check the context of this quotation to be sure
      of its accuracy.</p>
  </body>
  <context name="source">
    <agent role="author">
      <first>Jennifer</first>
      <middle>Alston</middle>
    </agent>
  </context>
</infoml>
```

```

    <middle>Middlesex</middle>
    <last>Hornsby</last>
    <suffix>Ph.D.</suffix>
    <contact-info>
      <email>jhornsby@udevanne.edu</email>
      <other name="aol">hornsby1029973</other>
    </contact-info>
  </agent>
  <date-created>2004-01-01</date-created>
  <container style="APA">
    <name-part separator=":">Beyond the Garden of Eden</name-part>
    <name-part>Therapy with Children and Their Families</name-part>
    <name type="publisher">Norton-Wise Publications</name>
    <date type="publication">1997</date>
    <contact-info>
      <local-address>1234 Andrea St.</local-address>
      <local-address>Building B</local-address>
      <city>Santo Domingo</city>
      <state-or-province>California</state-or-province>
      <postal-code>98977</postal-code>
      <country>USA</country>
      <email>director@norton-professional.com</email>
      <phone>1-408-555-1234</phone>
      <other name="aol">nordir</other>
    </contact-info>
  </container>
  <location style="APA">
    <id-part unit="volume">3</id-part>
    <id-part unit="issue">2</id-part>
    <range unit="page">
      <begin>114</begin>
      <end>117</end>
    </range>
    <point unit="page">119</point>
  </location>
</context>
<context name="this-card">
  <agent role="creator">
    <first>Gregory</first>
    <middle>P.</middle>
    <last>Williams</last>
    <contact-info>
      <email>gw227@scu.edu</email>
    </contact-info>
  </agent>
  <date-created>none</date-created>
  <date-modified>2004-01-07T11:15:04</date-modified>
  <date-modified>2004-02-08T17:52:42</date-modified>
</context>
<comment-on>
  <ptr>
    <card-id>gw667_144</card-id>
    <pnotes>
      <p>check this reference first</p>
    </pnotes>
  </ptr>
  <ptr>
    <card-id>pat136.aol.com_1105</card-id>
  </ptr>
</comment-on>
<pointers name="descending-list" quality="importance" notes="other research su
  <ptr>
    <card-id>gw667_103</card-id>
    <pnotes>

```

```
        <p>major point to be made</p>
      </pnotes>
    </ptr>
  <ptr>
    <card-id>gw667_104</card-id>
    <pnotes>
      <p>minor point</p>
    </pnotes>
  </ptr>
</pointers>
</infoml>
```

---

# Chapter 3. Customizing Infocards

## 3.1. Levels of Customization

One measure of the usefulness of a standard is its extensibility—that is, how much it can be modified to meet new needs. Excluding the extensibility that comes from revising the InfoML Standard itself, there are four levels of customization for information captured in the InfoML format:

1. **Standard infocards:** These contain nothing outside the InfoML Standard. All the information in such an infocard must be displayed by any InfoML-friendly application.
2. **User-customized infocards:** These include information in the `body//notes` or `special//this-card` fields. All the information in such an infocard must be displayed by any InfoML-friendly application.
3. **Enhanced infocards:** These contain developer-specific elements (marked by their name attribute being an IRI string). InfoML-friendly applications may not display developer-specific elements, but they will also not discard them. Such infocards require special applications to display their developer-specific elements, and they can only display the developer-specific elements they were designed to recognize.
4. **Custom infocards:** These conform to the Level 1 syntax of the InfoML Standard but are free to disregard any of the Level 2 rules. Such infocards cannot be read by InfoML-friendly applications, but the developer can make use of basic InfoML tools and selected APIs to make development easier. Custom infocards are defined by a `<selector>` element that has an IRI string as the value of its name attribute.

## 3.2. Application Handling of Customized Infocards

It is important that any InfoML-friendly application not discard any developer-specific elements in an infocard. If this convention is followed, information in the developer-specific elements will behave like sticky notes on a conventional notecard—even when the “notecards” are rearranged or modified (a set of InfoML data being manipulated by some InfoML-friendly application), the sticky notes will not “fall off.”

Apart from the stipulation that InfoML-friendly applications not discard developer-specific elements, application developers still have certain decisions to make:

- **Developer-specific elements:** In some situations, developers may want to enable their users to have access to developer-specific elements, displaying their contents as raw data. Usually, however, it will not make sense for a generic InfoML-friendly application to attempt to display such elements.
- **Invalid infocards:** An infocard may be invalid on either the syntactic (Level 1) or in the semantic (Level 2) level. Developers will need to decide how their InfoML-friendly applications will handle invalid infocards. Possible responses include: to stop processing infocards immediately; to stop processing the current infocard only, while bringing the error to the user’s attention in some way; or to give the user the opportunity to correct the invalid infocard, then resume processing.

---

# Chapter 4. Design and Implementation Issues

This chapter discusses issues of interest to developers who are considering using InfoML.

## 4.1. Adding InfoML to your Application

One of the goals of InfoML is to serve as a standard format for the storage, sharing, and manipulation of information. The more applications that understand information stored in the InfoML format, the more useful such stored information will be. In addition, once multiple applications support InfoML Standard, applications within a given product category that support InfoML will be seen as more useful than applications that do not.

If you have not finished your application, you may want to consider using InfoML as the native standard for storing your application's documents. If you have an existing application, it may be sufficient simply to make InfoML as one of the data formats your application can read and write (i.e., import and export functions).

## 4.2. Minimizing the Use of Private Data

If your application is the only one that reads the data you have stored in the InfoML format, you do not need to be overly concerned about how you map your data structures to the elements and attributes that InfoML provides. In particular, one quick solution is to create developer-specific elements (i.e., elements that are not visible to an InfoML-friendly application) that make the mapping of your data structures to InfoML straightforward.

However, this solution undermines InfoML's stated goal of providing a format that enables different kinds of applications to exchange data freely. Even if you are designing an in-house application that (at least, at the time of design) doesn't need to exchange data with other applications, you should still consider using a data mapping that maximizes the potential for data interchange in the future.

The key idea here is to find ways to place your data in standard InfoML elements and, wherever possible, to annotate that data in a way that helps users understand its purpose. There are different ways of doing this; the example below describes one such way.

Assume that you, as an Infocorp employee, have developed a simple PIM (personal information manager) that includes a to-do list and an appointment book, each of which has data structures associated with individual records.

Your first thought might be to create a developer-specific `<selector>` element to mark, for example, certain infocards as appointment items, like this:

```
<selector name="infocorp.com_appt">high-priority</selector>
```

A better solution would be to use a standard InfoML keyword instead:

```
<selector name="key">infocorp.com_appointment high-priority</selector>
```

InfoML-friendly applications would not see the data in the former solution, but it would in the latter; furthermore, the string "infocorp.com\_appointment" would give the reader some indication of the infocard's meaning.

## 4.3. Standard Locations for Information

There is only one appropriate place for the most important information associated with an infocard, and that is the `body//source` element. However, there are other places you can add information to a standard infocard (that is, one that does not include developer-specific elements).

### 4.3.1. Standard Locations for “Visible” Information

In addition to the `body//source` element, there are two other locations for “visible” information (i.e., information that should be visible using most, if not all, InfoML-friendly applications). The first is the `body//notes` element, which every InfoML-friendly application should make visible. This element is simply a text string that can contain any text.

The second such location is the `special//source` element, which can store a variety of data structures; see section 2.5.9, “The `<special>` Element.”

### 4.3.2. Standard Locations for “Invisible” Information

InfoML stores data of interest to the user in elements and all other data (“invisible” data or metadata) in element attributes. All the InfoML top-level subelements, as well as the `<infoml>` and `<infoml-file>` elements, contain two optional attributes, `custom1` and `custom2`, that developers can use however they wish for data the user does not need to see.

---

# Appendix A. The InfoML Resource Indicator (IRI) String

The IRI string is a very important component of the InfoML Standard. Its main characteristic is that it is *globally unique*—that is, whenever a new IRI string is created, it is guaranteed that no one else in the world is using the same string.

IRI strings are used for several purposes:

- as a string that uniquely identifies a specific infocard (by using it as the value of the infocard’s `<cid>` element);
- as a string that a developer uses to create a developer-specific element for use in an enhanced infocard; this is done by setting the element’s name attribute’s value to the IRI string;
- as part of a keyword that takes the place of a developer-specific `<selector>` element (see section 4.2, “Minimizing the Use of Private Data”).

## A.1. Some Examples

The following is an informal description of valid IRI strings; for their formal definition, see the definition of `cidType` in Appendix B.

IRI strings are globally unique because of an included substring that “belongs” to the person who creates it. Here are three canonical examples:

1. “pat.infocorp.com\_117” (used if the creator controls the e-mail address pat@infocorp.com);
2. “infocorp.com\_117” (used if the creator controls the domain infocorp.com);
3. “pat79\_117” (used if the creator controls the e-mail address pat79@yahoo.com).

Relating to the third example, the domain yahoo.com was chosen because it offers free e-mail accounts worldwide.

## A.2. Terminology, Rules, and Usage

In an IRI string, the IRI global part is the first part of the string, up to but not including the first underscore (“\_”) character. To increase future extensibility, an IRI string can include up to two underscore characters—for example, “pat.infocorp.com\_117\_a1”.

The entire substring before the first underscore (in the last example, “pat.infocorp.com”) is called the *IRI global part*. The method described earlier for creating the IRI global part ensures that anyone to who wants to create an infocard can do so. The IRI global part can contain letters, numbers, the period character (“.”), and the hyphen character (“-”). It must also begin with a letter; this means that e-mail addresses and domain names that begin with a number cannot be used to create an IRI global part.

Similarly, the substring after the first underscore (“117\_a1”) is called the *IRI local part*. This substring can contain any number of letters, numbers, the period character (“.”), and the hyphen character (“-”); in addition, it may contain no more than one underscore (“\_”) character. Note that the second underscore

makes it easy for a developer to consider the IRI local part as being two separate values separated by an underscore character (e.g., “location\_lowerleftquad”).

The content of the IRI local part is entirely up to the string’s creator, as long as the entire IRI string conforms to the rules given so far. In practice, only the developer of an InfoML-friendly application will need to design a method for generating IRI local-part strings; end-users should be responsible only for entering an IRI global-part string based on an e-mail address or domain name.

Since the IRI global part remains constant for a given creator, it is the responsibility of the IRI local part to make the entire IRI string globally unique. Two strategies for doing so are using consecutive values (e.g., “117”, “118”, and so on) and using strings guaranteed to be unique (e.g., making the IRI local part equal to some hash value computed from the content of the `body//source` element). Of course, creators are free to use other methods that result in globally unique IRI strings.

---

# Appendix B. The InfoML 0.83 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- -->
  <!-- -->
  <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
  <!--           specialized basic data types           -->
  <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
  <!-- -->
  <xsd:simpleType name="nonnullTokenType">
    <xsd:annotation>
      <xsd:documentation>Surprisingly, an empty string is a valid token.
        This type is used for tokens that must have at least one
        character in them.</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:token">
      <xsd:minLength value="1"/>
    </xsd:restriction>
  </xsd:simpleType>
  <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
  <xsd:complexType name="nameAttrNonnullTokenType">
    <xsd:annotation>
      <xsd:documentation>This type is used for non-null tokens that have a
        required 'name' attribute.</xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
      <xsd:extension base="nonnullTokenType">
        <xsd:attribute name="name" type="nonnullTokenType"
          use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
  <xsd:complexType name="attrCidType">
    <xsd:annotation>
      <xsd:documentation>This type is used for non-null tokens that have
        and optional 'custom1' and 'custom2' attributes, the value of
        which must be a cid.</xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
      <xsd:extension base="cidType">
        <xsd:attribute name="custom1" type="xsd:string" use="optional"/>
        <xsd:attribute name="custom2" type="xsd:string" use="optional"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
  <xsd:complexType name="typeAttrNonnullTokenType">
    <xsd:annotation>
      <xsd:documentation>This type is used for non-null tokens that have a
        required 'type' attribute.</xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
      <xsd:extension base="nonnullTokenType">
        <xsd:attribute name="type" type="nonnullTokenType"
          use="required"/>
        <xsd:attribute name="custom1" type="xsd:string" use="optional"/>
        <xsd:attribute name="custom2" type="xsd:string" use="optional"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:schema>
```





```

        maxOccurs="unbounded" type="nonnullTokenType"/>
<xsd:element name="postal-code" minOccurs="0" maxOccurs="unbounded"
  type="nonnullTokenType"/>
<xsd:element name="country" minOccurs="0" maxOccurs="unbounded"
  type="nonnullTokenType"/>
<xsd:element name="email" minOccurs="0" maxOccurs="unbounded"
  type="nonnullTokenType"/>
<xsd:element name="phone" minOccurs="0" maxOccurs="unbounded"
  type="nonnullTokenType"/>
<xsd:element name="other" minOccurs="0" maxOccurs="unbounded"
  type="nameAttrNonnullTokenType"/>
</xsd:sequence>
</xsd:complexType>
<!-- -->
<xsd:complexType name="agentType">
  <xsd:sequence>
    <xsd:element name="prefix" minOccurs="0" maxOccurs="unbounded"
      type="nonnullTokenType"/>
    <xsd:element name="first" minOccurs="0" type="nonnullTokenType"/>
    <xsd:element name="middle" minOccurs="0" maxOccurs="unbounded"
      type="nonnullTokenType"/>
    <xsd:element name="last" minOccurs="0" type="nonnullTokenType"/>
    <xsd:element name="suffix" minOccurs="0" maxOccurs="unbounded"
      type="nonnullTokenType"/>
    <xsd:element name="contact-info" minOccurs="0"
      type="contactInfoType"/>
  </xsd:sequence>
  <xsd:attribute name="role" type="nonnullTokenType"/>
</xsd:complexType>
<!-- -->
<!-- -->
<xsd:complexType name="titlePartType">
  <xsd:simpleContent>
    <xsd:extension base="nonnullTokenType">
      <xsd:attribute name="separator" type="nonnullTokenType"
        use="optional"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<!-- -->
<xsd:complexType name="containerType">
  <xsd:sequence>
    <xsd:element name="name-part" minOccurs="0" maxOccurs="unbounded"
      type="titlePartType"/>
    <xsd:element name="name" minOccurs="0" maxOccurs="unbounded"
      type="typeAttrNonnullTokenType"/>
    <xsd:element name="date" minOccurs="0" maxOccurs="unbounded"
      type="typeAttrNonnullTokenType"/>
    <xsd:element name="contact-info" minOccurs="0"
      type="contactInfoType"/>
  </xsd:sequence>
  <xsd:attribute name="type" use="optional"/>
  <xsd:attribute name="style" use="optional"/>
</xsd:complexType>
<!-- -->
<!-- -->
<xsd:complexType name="idPartType">
  <xsd:simpleContent>
    <xsd:extension base="nonnullTokenType">
      <xsd:attribute name="unit" type="nonnullTokenType"
        use="optional"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

```

```

<!-- -->
<xsd:complexType name="rangeType">
  <xsd:sequence>
    <xsd:element name="begin" type="nonnullTokenType"/>
    <xsd:element name="end" type="nonnullTokenType"/>
  </xsd:sequence>
  <xsd:attribute name="unit" type="nonnullTokenType" use="optional"/>
</xsd:complexType>
<!-- -->
<xsd:complexType name="pointType">
  <xsd:simpleContent>
    <xsd:extension base="nonnullTokenType">
      <xsd:attribute name="unit" type="nonnullTokenType"
        use="optional"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<!-- -->
<xsd:complexType name="locationType">
  <xsd:sequence>
    <xsd:element name="id-part" minOccurs="0" maxOccurs="unbounded"
      type="idPartType"/>
    <xsd:choice maxOccurs="unbounded">
      <xsd:element name="range" type="rangeType"/>
      <xsd:element name="point" type="pointType"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="style" type="nonnullTokenType" use="optional"/>
</xsd:complexType>
<!-- -->
<xsd:complexType name="contextType">
  <xsd:sequence>
    <xsd:element name="agent" minOccurs="0" maxOccurs="
      unbounded" type="agentType"/>
    <xsd:element name="date-created" minOccurs="0" type="dateType"/>
    <xsd:element name="date-modified" minOccurs="0"
      maxOccurs="unbounded" type="dateType"/>
    <xsd:element name="container" minOccurs="0" maxOccurs="
      unbounded" type="containerType"/>
    <xsd:element name="location" minOccurs="0" type="locationType"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="nonnullTokenType" use="required"/>
  <xsd:attribute name="custom1" type="xsd:string" use="optional"/>
  <xsd:attribute name="custom2" type="xsd:string" use="optional"/>
</xsd:complexType>
<!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
<!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
<!-- pointersType -->
<!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
<xsd:complexType name="ptrType">
  <xsd:sequence>
    <xsd:element name="card-id" type="cidType"/>
    <xsd:element name="pnotes" minOccurs="0" type="bodyContents"/>
  </xsd:sequence>
</xsd:complexType>
<!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
<xsd:complexType name="pointersType">
  <xsd:sequence maxOccurs="unbounded">
    <xsd:element name="ptr" type="ptrType"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="nonnullTokenType" use="required"/>
  <xsd:attribute name="quality" type="nonnullTokenType" use="optional"/>
  <xsd:attribute name="notes" type="nonnullTokenType" use="optional"/>

```

```

        <xsd:attribute name="custom1" type="xsd:string" use="optional"/>
        <xsd:attribute name="custom2" type="xsd:string" use="optional"/>
    </xsd:complexType>
    <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
    <xsd:complexType name="commentOnType">
        <xsd:sequence maxOccurs="unbounded">
            <xsd:element name="ptr" type="ptrType"/>
        </xsd:sequence>
        <xsd:attribute name="notes" type="nonnullTokenType" use="optional"/>
        <xsd:attribute name="custom1" type="xsd:string" use="optional"/>
        <xsd:attribute name="custom2" type="xsd:string" use="optional"/>
    </xsd:complexType>
    <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
    <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
    <!-- specialType -->
    <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
    <xsd:complexType name="specialType">
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="name" type="nonnullTokenType"
                    use="required"/>
                <xsd:attribute name="major-delimiter" type="nonnullTokenType"
                    use="optional"/>
                <xsd:attribute name="minor-delimiter" type="nonnullTokenType"
                    use="optional"/>
                <xsd:attribute name="tertiary-delimiter" type="nonnullTokenType"
                    use="optional"/>
                <xsd:attribute name="table-type" type="nonnullTokenType"
                    use="optional"/>
                <xsd:attribute name="custom1" type="xsd:string" use="optional"/>
                <xsd:attribute name="custom2" type="xsd:string" use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
    <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
    <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
    <!-- infoml element -->
    <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
    <xsd:simpleType name="versionType">
        <xsd:restriction base="xsd:decimal">
            <xsd:minInclusive value="0.83"/>
            <xsd:maxInclusive value="0.83"/>
        </xsd:restriction>
    </xsd:simpleType>
    <!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
    <xsd:complexType name="infomlType">
        <xsd:sequence>
            <xsd:element name="cid" type="attrCidType"/>
            <xsd:element name="pid" minOccurs="0" maxOccurs="
                unbounded"
                type="nameAttrNonNullTokenType"/>
            <xsd:element name="selector" minOccurs="0" maxOccurs="
                unbounded"
                type="nameAttrNonNullTokenType"/>
            <xsd:element name="tag" minOccurs="0" maxOccurs="
                unbounded"
                type="nameAttrNonNullTokenType"/>
            <xsd:element name="body" maxOccurs="unbounded" type="
                bodyType"/>
            <xsd:element name="context" minOccurs="0" maxOccurs="
                unbounded"
                type="contextType"/>
            <xsd:element name="comment-on" minOccurs="0" type="
                commentOnType"/>
        </xsd:sequence>
    </xsd:complexType>

```

```

    <xsd:element name="pointers" minOccurs="0" maxOccurs=
      "unbounded"
      type="pointersType"/>
    <xsd:element name="special" minOccurs="0" maxOccurs=
      "unbounded"
      type="specialType"/>
  </xsd:sequence>
  <xsd:attribute name="signature" type="xsd:string" use="optional"/>
  <xsd:attribute name="encoding" type="nonnullTokenType"
    use="required"/>
  <xsd:attribute name="version" type="versionType" use="required"/>
  <xsd:attribute name="based-on" type="cidType" use="optional"/>
  <xsd:attribute name="custom1" type="xsd:string" use="optional"/>
  <xsd:attribute name="custom2" type="xsd:string" use="optional"/>
</xsd:complexType>
<!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
<!-- infoml-file -->
<!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
<xsd:element name="infoml-file">
  <xsd:complexType>
    <xsd:sequence maxOccurs="unbounded">
      <xsd:element name="infoml" type="infomlType"/>
    </xsd:sequence>
    <xsd:attribute name="title" type="nonnullTokenType" use="optional"/>
    <xsd:attribute name="custom1" type="xsd:string" use="optional"/>
    <xsd:attribute name="custom2" type="xsd:string" use="optional"/>
  </xsd:complexType>
</xsd:element>
<!-- ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ -->
</xsd:schema>
<!-- -->
<!--
CHANGE LOG:

0.83: Major restructuring, including: new comments element; combined
the group and content standard infocards into one kind of infocard;
added contactInfoType; added features to agentType, containerType,
locationType, and contextType; converted many global elements to
complexTypes, leaving infoml-file as the only global element.

0.80: Changed the author element to an agent element; added type attribute
to agent element; removed more-authors element in favor of multiple
agent elements.

-->

```

---

# Appendix C. Rules for the InfoML 0.83 Semantics (Level 2)

This appendix contains the Level 2 rules that any InfoML-related software must implement. Table 3, “Regular Rules for Level 2 of the InfoML Specification,” below, describes the “regular” rules—that is, the ones easily described by a set of similar relationships.

**Table C.1. Regular Rules for Level 2 of the InfoML Specification**

Element-name//name-attribute-value	Min # of occurrences	Max # of occurrences	See irregular rules?
selector//cardtype	1	1	rule 1
tag//title	0	1	rule 3
body//source	1	1	no
body//notes	0	1	no
context//notes	0	1	
context//this-card	0	1	no
pointers//some-IRI-string	0	unlimited	no

Unfortunately, there are exceptions to the regular rules, as well as other needed rules that are not so easily expressed. These comprise the “irregular rules,” which are listed as follows:

1. The value of the `selector//cardtype` element must be one of the following strings: “idea”, “opinion”, “fact”, “definition”, “narrative”, or “generic”. If the value of this element is an IRI string, then the infocard is a *custom infocard*, which is not bound by the Level 2 rules of the InfoML Specification.
2. The `body//source` element must occur exactly once in an infocard (though it may be an empty element).
3. If the `selector//cardtype` element has the value “definition”, then the `tag//title` element must exist. (In other words, a definition infocard must have a title.)
4. If a `context//original` element exists, then at least one `context//source` element must exist.
5. If a `context//middle` element exists, then at least one `context//original` element and at least one `context//source` element must exist.
6. If a `context//X` element exists, where X = “source”, “notes”, or an IRI string, then the corresponding `body//X` element must exist. (Note that, although `context//middle` and `context//original` elements are valid elements, `body//middle` and `body//original` elements are not legal in an infocard.)
7. If a `context//original` element exists, then at least one `context//source` element must exist.

For details on the situation that necessitates rules 3 through 5, see section 2.5.6.4, “The `context//source`,

context//middle, and context//original Elements.”

# Appendix D. Important Elements and Attributes for InfoML 0.83

This appendix lists all the reserved values specified by the InfoML Specification, version 0.83, as well as other important values for selected elements and attributes. In the notation used, the “@” character refers to an identifier that is an attribute—for example, “selector/@name” refers to the name attribute of the <selector> element. In the values column, double parentheses indicate a description of the allowed value, as opposed to an actual string value.

**Table D.1. Reserved values for InfoML 0.83**

<b>Element or @attribute</b>	<b>Reserved value(s) or ((other value))</b>
agent/@role	author editor speaker
body/@name	source notes
cid	((an IRI string))
container/name-part/@separator	((When a container’s name is in multiple parts (e.g., book title with subtitle), each part of the name has its own name-part element. The value of the separator attribute is a string meant to be used after the current name-part and before the next name-part.))
context/@name	source middle original
infoml/@based-on	((the cid value of the infocard upon which the current infocard is based))
location/id-part/@unit	book chapter issue page volume
pid/@name	((any non-empty string))
pointers/@name	source
selector/@name, where name = “key”	((any non-empty string))
selector/@name, where name = “cardtype”	definition fact generic idea narrative opinion ((an IRI string))
special/@name	source
special/@table-type	2d-array 3d-array irreg-prop-list list prop-list
tag/@name	title descriptor