Ink Markup Language (InkML)

W3C Working Draft 23 October 2006

This version:
http://www.w3.org/TR/2006/WD-InkML-20061023
Latest version:
http://www.w3.org/TR/InkML
Previous version:
http://www.w3.org/TR/2004/WD-InkML-20040928
Editors:
Yi-Min Chee, IBM
Max Froumentin, W3C
Stephen M. Watt, University of Western Ontario
Authors:
Yi-Min Chee, IBM
Katrin Franke, Fraunhofer Gesellschaft
Max Froumentin, W3C
Sriganesh Madhvanath, HP
Jose-Antonio Magaña, HP
Gregory Russell, IBM
Giovanni Seni, Motorola
Christopher Tremblay, Corel
Stephen M. Watt, University of Western Ontario
Larry Yaeger, Apple

A non-normative version of this document showing changes made since the previous draft is also available.

Abstract

This document describes the syntax and semantics for the Ink Markup Language for use in the W3C Multimodal Interaction Framework as proposed by the W3C Multimodal Interaction Activity. The Ink Markup Language serves as the data format for representing ink entered with an electronic pen or stylus. The markup allows for the input and processing of handwriting, gestures, sketches, music and other notational languages in applications. It provides a common format for the exchange of ink data between components such as handwriting and gesture recognizers, signature verifiers, and other ink-aware modules.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

The (archived) public mailing list www-multimodal@w3.org (see instructions) is preferred for discussion of this specification. When sending e-mail, please put the text "[ink]" in the subject, preferably like this: "[ink] *…summary of comment…*"

This document was produced by the Multimodal Interaction Working Group *(W3C Members Only)*, which is part of the Multimodal Interaction Activity.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document contains the InkML W3C Last Call Working Draft of 23 October 2006. The Last Call period ends on 18 December 2006.

This fourth version of the Working Draft includes a few conceptual changes to simplify the definition while achieving greater expressive power. It also contains many small changes of details to make element and attribute use uniform accross the the definition to make it easier to learn and simpler to process.

The main changes are:

- InkML now more robustly supports program transformations. The text has been revised to remove any requirement for a particular element order in archival ink. This allows applications to regroup and organize traces into logical structures without losing information.
- InkML now more robustly supports streaming. The content model of the top-level ink element has been relaxed to allow interspersion of more definitional elements. The definition of continuation traces has been simplified.
- InkML now better supports optical devices and other technologies. The language has been revised to be technology neutral, where possible, and to keep technology-specific concepts localized to specific elements.
- There is greater support for applications to use InkML as a representation for their own application-defined structures. Trace groups and trace references can be nested, allowing applications to group ink into logical units, if desired. This may be done to the explicit ink traces or by reference.
- The support for annotation has been enhanced to allow arbitrary textual or XML-based annotation. This provides sufficient hooks for rich semantic annotation of ink while keeping the standard simple. The model is based on experience with MathML.
- The concepts of trace formats and capture devices have been more clearly distinguished. Trace formats can be used to describe all the logical properties of an ideal channel. They are used to describe traces and the coordinates of shared

canvases. Consequently, the channel element has a richer set of attributes. Capture devices are now seen as "ink sources" which may additionally describe other characteristics of the ink source, such as accuracy, latency, channel cross-coupling, etc.

- The notions of canvas transformations and channel mappings have been converged into a single mapping type. As a consequence, applications may agree on more general coordinate systems for shared canvases. (For example, they may share tip force information.)

Several changes of detail have been made to support the above, to make the naming and use of elements and attributes consistent, and to remove duplication.

For the full list of changes, see the appendix Changes from Previous Working Draft.

Table of Contents

---

1 Overview

As more electronic devices with pen interfaces have and continue to become available for entering and manipulating information, applications need to be more effective at using this method of input. Handwriting is a powerful and versatile input modality that is very familiar for most users since everyone learns to write in school. Hence, users will tend to use this as a mode of input and control when available.

A pen-based interface is enabled by a device that allows movements of the pen to be captured as digital ink. A number of methods may be used for ink capture, including those based on radio frequency, optical tracking, physical pressure, or other technologies. Digital ink can be passed on to recognition software that will convert the pen input into appropriate computer actions. Alternatively, the handwritten input can be organized into ink documents, notes or messages that can be stored for later retrieval or exchanged through telecommunications means. Such ink documents are appealing because they capture information as the user composed it, including text in any mix of languages and drawings such as equations and graphs.

Hardware and software vendors have typically stored and represented digital ink using proprietary or restrictive formats. The lack of a public and comprehensive digital ink format has severely limited the capture, transmission, processing, and presentation of digital ink across heterogeneous devices developed by multiple vendors. In response to this need, the Ink Markup Language (InkML) provides a simple and platform-neutral data format to promote the interchange of digital ink between software applications.

InkML supports a complete and accurate representation of digital ink. In addition to the pen position over time, InkML allows recording of information about device characteristics and detailed dynamic behavior to support applications such as handwriting recognition and authentication. For example, there is support to record additional information such as pen tilt and pen tip force (often referred to as "pressure") and information about the recording device such as accuracy and dynamic distortion. InkML also provides features to support rendering of digital ink captured optically to approximate the original appearance. For example, stroke width and color information can be recorded.

InkML provides means for extension. By virtue of being an XML-based language, users may easily add application-specific information to ink files to suit the needs of the application at hand.

1.1 Uses of InkML

With the establishment of a non-proprietary ink standard, a number of applications, old and new, are expanded where the pen can be used as a very convenient and natural form of input. Here are a few examples.

- **Ink Messaging**

  Two-way transmission of digital ink, possibly wireless, offers mobile-device users a compelling new way to communicate. Users can draw or write with a pen on the device's screen to compose a note in their own handwriting. Such an ink note can then be addressed and delivered to other mobile users, desktop users, or fax machines. The recipient views the message as the sender composed it, including text in any mix of languages and drawings.

- **Ink and SMIL**

  A photo taken with a digital camera can be annotated with a pen; the digital ink can be coordinated with a spoken commentary. The ink annotation could be used for indexing the photo (for example, one could assign different handwritten glyphs to different categories of pictures).

- **Ink Archiving and Retrieval**

  A software application may allow users to archive handwritten notes and retrieve them using either the time of creation of the handwritten notes or the tags associated with keywords. The tags are typically text strings created using a handwriting recognition system.

- **Electronic Form-Filling**

  In support of natural and robust data entry for electronic forms on a wide spectrum of keyboardless devices, a handwriting recognition engine developer may define an API that takes InkML as input.

- **Pen Input and Multimodal Systems**

  Robust and flexible user interfaces can be created that integrate the pen with other input modalities such as speech. Higher robustness is achievable because cross-modal redundancy can be used to compensate for imperfect recognition on each individual mode. Higher flexibility is possible because users can choose the most appropriate from among various modes for achieving a task or issuing

commands. This choice might be based on user preferences, suitability for the task, or external conditions. For instance, when noise in the environment or privacy is a concern, the pen modality is preferred over voice.

1.2 Elements

The current InkML specification defines a set of primitive elements sufficient for all basic ink applications. All content of an InkML document is contained within a single **<ink>** element. The fundamental data element in an InkML file is the **<trace>**. A trace represents a sequence of contiguous ink points, where each point captures the values of particular quantities such as the X and Y coordinates of the pen's position. A sequence of traces accumulates to meaningful units, such as characters, words or diagrams. The

In its simplest form, an InkML file with its enclosed traces looks like this:

```
<ink>
   <trace>
     10 0, 9 14, 8 28, 7 42, 6 56, 6 70, 8 84, 8 98, 8 112, 9 126, 10 140,
     13 154, 14 168, 17 182, 18 188, 23 174, 30 160, 38 147, 49 135,
     58 124, 72 121, 77 135, 80 149, 82 163, 84 177, 87 191, 93 205
   </trace>
   <trace>
     130 155, 144 159, 158 160, 170 154, 179 143, 179 129, 166 125,
     152 128, 140 136, 131 149, 126 163, 124 177, 128 190, 137 200,
     150 208, 163 210, 178 208, 192 201, 205 192, 214 180
   </trace>
   <trace>
     227 50, 226 64, 225 78, 227 92, 228 106, 228 120, 229 134,
     230 148, 234 162, 235 176, 238 190, 241 204
   </trace>
   <trace>
     282 45, 281 59, 284 73, 285 87, 287 101, 288 115, 290 129,
     291 143, 294 157, 294 171, 294 185, 296 199, 300 213
   </trace>
   <trace>
     366 130, 359 143, 354 157, 349 171, 352 185, 359 197,
     371 204, 385 205, 398 202, 408 191, 413 177, 413 163,
     405 150, 392 143, 378 141, 365 150
   </trace>
</ink>
```

These traces consist simply of X and Y value pairs, and may look like this when rendered:

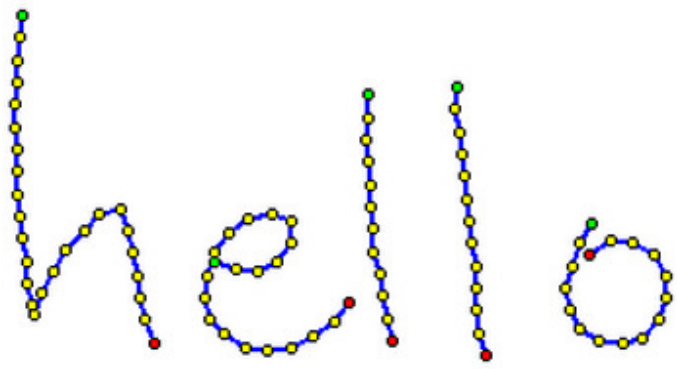

Figure 1: Example of trace rendering

Figure 1 shows a trace of a sampled handwriting signal. The dots mark the sampling positions which were interpolated by the blue line. Green points represent pen-downs whereas red dots indicate pen-ups.

More generally, traces consist of sequences of points. Each point consists of a number of coordinate values whose meanings are given by a **<traceFormat>** element. These coordinates may provide values for such quantities as pen position, angle, tip force, button states and so on.

Information about the device used to collect the ink (e.g., the sampling rate and resolution) may be specified with the **<inkSource>** element.

Ink traces can have certain attributes such as color and width, writer identification, pen modes (eraser vs writing), and so on. These and other attributes are captured using the **<brush>** element. Traces that share the same characteristics, such as being written with the same brush, can be grouped together with the **<traceGroup>** element.

Ink traces may also be organized into collections for application-specific purposes either by grouping the traces objects themselves, using the **<traceGroup>** element, or by reference, using the **<traceView>** element.

Certain applications, such as collaborative whiteboards (where ink coming from different devices is drawn on a common canvas) or document review (where ink annotation from various sources is combined), will require ink sharing. The **<context>** element allows representation and grouping of the pertinent information, such as the trace format, brush, and canvas. Canvas transformations allow ink from different devices to combined and manipulated by multiple parties.

InkML supports the semantic labelling of traces with attributes on traces or collections of traces. These may be given with either **<annotation>**, for text, or **<annotationXML>**, for XML, using application-defined encodings.

In all appropriate cases, the InkML specification defines default values for elements that are not specified, and rules that establish the scope of a given attribute.

Finally, the InkML specification is limited in scope: It is currently oriented to fixed Cartesian coordinate systems, it does not support sophisticated compression of trace data, and it does not support non-ink events (although the later could be handled via annotations).

1.3 Exchange Modes

Most ink-related applications fall into two broad categories: "Streaming" and "Archival". Archival ink applications capture and store digital ink for later processing, such as document storage/retrieval applications and batch forms processing . In these applications, an entire **<ink>** element is written prior to processing. For ease of implementation, it is recommended that, in archival mode, referenced elements be defined inside a declaration block using the **<definitions>** element.

Streaming ink applications, on the other hand, transmit digital ink as it is captured, such as in the electronic whiteboard example mentioned above. In order to support a streaming style of ink markup generation, the InkML language supports the notion of a "current" state (e.g., the current brush) and allows for incremental changes to this state.

1.4 Conventions used in this document

This document uses the following conventions:

Syntax of element contents

The syntax of the contents of InkML elements is expressed in Backus-Naur Form,

using the notation defined in the Trace section. Non-literal symbols represent InkML markup and are linked to the relevant section in this document. For example:

```
channel* intermittentChannels?
```

Syntax of attribute contents

In this specification attributes definitions are formatted as:

default = xsd:decimal | xsd:boolean

The lefthand side of the '=' sign is the name of the attribute and the right handside describes the syntax of the attribute's contents, using the same Backus-Naur Form notation as used for element definitions. In addition, a non-literal symbol will represent a data type name. By convention, this specification uses the prefix 'xsd:' to indicate that the following name is that of a datatype formally defined in the XML Schema Part 2: Datatypes Recommendation [*XMLSCHEMA2*]. The 'xsd' prefix is used only as a notation in this specification, and does not mandate any prefix when using XML Schema names in InkML.

---

2 Structure

InkML documents are well-formed XML documents which comply to the syntax rules of this specification.

The namespace URI of InkML is **`http://www.w3.org/2003/InkML`**

The media type of InkML document is **`application/inkml+xml`**. See the Media Type definition for details. This media type is expected to be registered with IETF.

2.1 `<ink>` element

The **`ink`** element is the root element of any InkML instance. When combining InkML and other XML elements within applications, elements from different namespaces must be disambiguated by use of the namespace qualifier. The allowed sub-elements of the **`ink`** element can occur any number of times, in any order.

ATTRIBUTES:

documentID = xsd:anyURI

A URI that uniquely identifies this document. No two documents with a distinct application intent may have the same **`documentID`** contents. The value of this property is an opaque URI whose interpretation is not defined in this specification.

Required: no, *Default:* none

CONTENTS:

```
( definitions | context | trace | traceGroup | traceView | annotation |
annotationXML )*
```

EXAMPLE:

```
<ink xmlns="http://www.w3.org/2003/InkML"
     documentID="uuid:6B29FC40-CA47-1067-B31D-00DD010662DA"/>
...
</ink>
```

---

3 Traces and Trace Formatting

Traces are the basic element used to record the trajectory of a pen as a user writes digital ink. More specifically, these recordings describe sequences of connected points. On most devices, these sequences of points will be bounded by pen contact change events (pen-up and pen-down), although some devices may simply record proximity and force data without providing an interpretation of pen-up or pen-down state.

The simplest form of encoding specifies the X and Y coordinates of each sample point. For compactness, it may be desirable to specify absolute coordinates only for the first point in the trace and use delta-x and delta-y values to encode subsequent points. Some devices record acceleration rather than absolute or relative position; some provide additional data that may be encoded in the trace, including Z coordinates or tip force, or the state of side switches or buttons.

These variations in the information available from different ink sources, or needed by different applications, are supported in InkML through the **`<traceFormat>`** and **`<trace>`** elements. The **`<traceFormat>`** element specifies the encoding format for each sample of a recorded trace, while **`<trace>`** elements are used to represent the actual trace data. If no **`<traceFormat>`** is specified, a default encoding format of X and Y coordinates is assumed.

Traces generated by different devices, or used in differing applications, may contain different types of information. InkML defines *channels* to describe the data that may be encoded in a trace.

A channel can be characterized as either *regular*, meaning that its value is recorded for every sample point of the trace, or *intermittent*, meaning that its value may change infrequently and thus will not necessarily be recorded for every sample point. X and Y coordinates are examples of likely regular channels, while the state of a pen button is likely to be an intermittent channel.

3.1 Trace Formats

3.1.1 **`<traceFormat>`** element

ATTRIBUTES:

xml:id = xsd:ID
The unique identifier for this trace format.
Required: no, *Default:* none

CONTENTS:

channel* intermittentChannels?

The **<traceFormat>** element describes the format used to encode points within **<trace>** elements. In particular, it defines the sequence of channel values that occurs within **<trace>** elements. The order of declaration of channels in the **<traceFormat>** element determines the order of appearance of their values within **<trace>** elements.

Regular channels appear first in the **<trace>**, followed by any intermittent channels. Correspondingly, the **<traceFormat>** element contains an ordered sequence of **<channel>**s, giving the regular channels (if any), followed by an optional **<intermittentChannels>** section. The order of the coordinates in each point of a trace is determined by the order of the **<channel>** elements in the trace format, including those from the intermittent channels part.

3.1.2 <intermittentChannels> element

ATTRIBUTES:

none

CONTENTS:

channel*

The **<intermittentChannels>** lists those channels whose value may optionally be recorded for each sample point. The order of the enclosed channel declarations gives the order of the intermittent channel data samples within traces having this format.

3.1.3 <channel> element

ATTRIBUTES:

xml:id = xsd:ID
The unique identifier for this element.
Required: no, *Default:* none

name = xsd:string
The name of this channel.
Required: yes

type = "integer" | "decimal" | "boolean"
The data type of the point values for this channel.
Required: no, *Default:* "decimal"

default = xsd:decimal | xsd:boolean

The default value of the point data for this channel. This only applies to intermittent channels.
Required: no, *Default:* 0 (for integer or decimal channel), F (for boolean channel)

min = xsd:number
The lower boundary for the values of this channel.
Required: no, *Default:* none

max = xsd:number
The upper boundary for the values of this channel.
Required: no, *Default:* none

orientation = "+ve" | "-ve"
The orientation of increasing channel values with respect to the default direction of the channel's coordinate axis, where applicable.
Required: no, *Default:* "+ve"

respectTo = xsd:anyURI
Specifies that the values are relative to another reference point.
Required: no, *Default:* none

units = xsd:string
The units in which the values of the channel are xpressed (numerical channels only).
Required: no, *Default:* none

CONTENTS:

`mapping?`

Channels are described using the `<channel>` element, with various attributes.

The required **name** attribute specifies the interpretation of the channel in the trace data. The following channel names, with their specified meanings, are reserved:

channel name	dimension	interpretation
X	length	X coordinate. This is the horizontal pen position on the writing surface, increasing to the right for +ve orientation.
Y	length	Y coordinate. This is the vertical position on the writing surface, increasing downward for +ve orientation.
Z	length	Z coordinate. This is the height of pen above the writing suface, increasing upward for +ve orientation.
F	force	pen tip force

S		tip switch state (touching/not touching the writing surface)
B1...Bn		side button states
OTx	angle	tilt along the x-axis
OTy	angle	tilt along the y-axis
OA	angle	azimuth angle of the pen (yaw)
OE	angle	elevation angle of the pen (pitch)
OR	angle	rotation (rotation about pen axis)
C		color value (device-specific encoding)
CR,CG,CB		color values (Red/Green/Blue)
CC,CM,CY,CK		color values (Cyan/Magenta/Yellow /Black)
W	length	stroke width (orthogonal to stroke)
T	time	time (of the sample point)

The **type** attribute defines the encoding type for the channel (either boolean, decimal, or integer). If **type** is not specified, it defaults to decimal.

A default value can be specified for the channel using the **default** attribute; the use of default values within a trace is described in the next section. If no **default** is specified, it is assumed to be zero for integer and decimal-valued channels, and false for boolean channels.

The **min** and **max** attributes, if given, specify the minimum and maximum possible values for a channel of type integer or decimal. If neither is given, then there is no a prior bound on the channel values. If one is given, then the channel values are bounded above or below but unbounded in the other direction. If both are given, then all channel values must fall within the specified range.

The **orientation** attribute is applicable to channels of integer or decimal type. It gives the meaning of increasing value. For example, whether X increases to the left or the right. The value may be given as "+ve" or "-ve", with "+ve" being the default.

The **respectTo** attribute specifies the origin for channels of integer or decimal type. For time channels, this is given as a URI for a `<timestamp>` element. For other dimensions the meaning is application-dependent.

Typically, a channel in the `<traceFormat>` will map directly to a corresponding channel provided by the digitizing device, and its values as recorded in the trace data will be the original channel values recorded by the device. However, for some applications, it may be useful to store normalized channel values instead, or even to remap the channels provided by the digitizing device to different channels in the trace data. This correspondence between the trace data and the device channels is recorded using a `<mapping>` element (described in the Mappings section) within the `<channel>` element. If no mapping is specified for a channel, it is assumed to be unknown.

3.1.4 Orientation Channels

The channels OTx, OTy, OA, OE and OR are defined for recording of pen orientation data. Implementers may choose to use either pen azimuth OA and pen elevation OE, or alternatively tilt angles OTx and OTy. The latter are the angles of projections of the pen

axis onto the XZ and YZ planes, measured from the vertical. It is often useful to record the sine of this angle, rather than the angle itself, as this is usually more useful in calculations involving angles. The **<mapping>** element can be employed to specify an applied sine transformation.

The third degree of freedom in orientation is generally defined as the rotation of the pen about its axis. This is potentially useful (in combination with tilt) in application such as illustration or calligraphy, and signature verification.

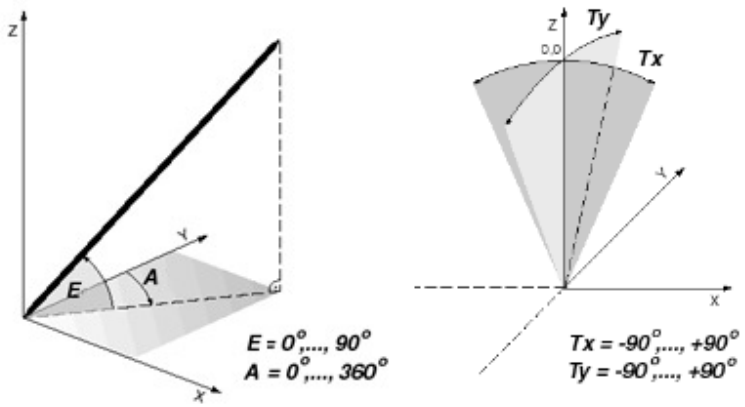

Figure 2: (a) azimuth and elevation angles, (b) tilt angles

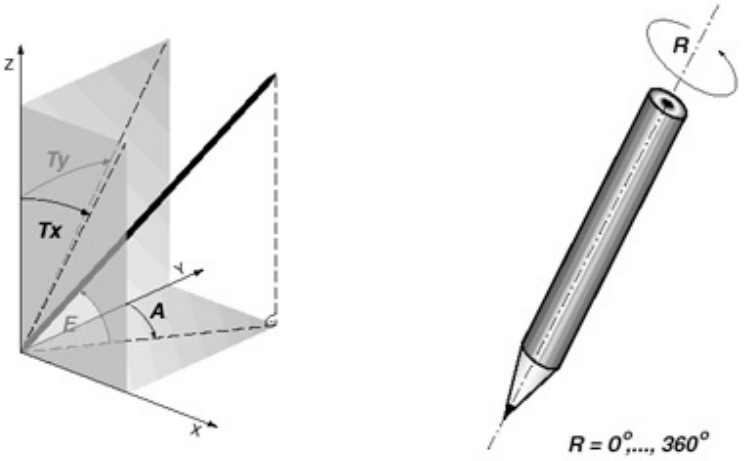

Figure 3: (a) pen orientation decomposition, (b) pen rotation

Figure 2a displays the pen orientation using Azimuth and Elevation. The origin of the Azimuth is at the Y-axis. Azimuth increases anticlockwise up to 360 degrees. The origin of Elevation is located within the XY-plane. Elevation increases up to 90 degrees, at which point the pen is perpendicular to the XY-plane.

Figure 2b explains the definition of the Tilt-X and the Tilt-Y angles. For both the origin is along the Z-axis. Tilt-X increases up to +90 degrees for inclinations along the positive X-axis and decreases up to -90 degrees for inclinations along the negative X-axis. Respectively, Tilt-Y is defined for pen inclinations along the Y-axis.

Figure 3a displays the pen orientation decomposition as functions of Azimuth/Elevation or alternatively as function of Tilt-X/Tilt-Y. Thereby, Elevations of the pen which are mapped to the XZ- and to the YZ- plane lead to Tilt-X and Tilt-Y.

Figure 3b shows the Rotation of the pen along its longitudinal axis.

3.1.5 Color Channels

The channels CR, CG, CB, CC, CM, CY, CK and C are defined to record color data as captured by an optical device, software settings or other means.

The channels CR, CG, CB provide an additive color model for the colors red, green and blue. The channels CC, CY, CM, CK provide a subtractive color model for the colors cyan, magenta, yellow and black. The channel C provides a mechanism to give color as a single numerical value, e.g. as a gray scale or a device-dependent hex-encoded number.

Color channels are intended for use when these values are part of the data itself and hence potentially changing from one sample to the next. Strokes with constant color may more economically be described with reference to a **<brush>** element.

3.1.6 Width Channel

The channel W is provided for recording stroke width. This allows optical devices to record measured stroke width and allows applications that generate InkML to specify desired width for rendering. The value is in length units, measured orthogonally to the stroke direction.

As with the color channels, the width channel is intended for use when this quantity is part of the data itself and hence potentially changing from one sample to the next. Strokes with constant width may more economically be described with reference to a **<brush>** element.

3.1.7 Time Channel

The time channel allows for detailed recording of the timing information for each sample point within a trace. This can be useful if the digitizing device has a non-uniform sampling rate, for example, or in cases where duplicate point data is removed for the sake of compactness.

The time channel can be specified as either a regular or intermittent channel. When specified as a regular channel, the single quote prefix can be used to record incremental time between successive points. Otherwise, the value of the time channel for a given sample point is defined to be the timestamp of that point in the units and frame of reference specified by its corresponding **<inkSource>** description (more precisely, by the **<traceFormat>** element for the channel).

As with the other predefined channels, the meaning of the integer or decimal values recorded by the time channel in a given trace is defined by the **<inkSource>** information associated with the trace's traceFormat. In the case of the time channel, its <channel> element contains both a **units** and **respectTo** attribute.

The **units** attribute gives the units of the recorded time values, and the **relativeTo** attribute describes the frame of reference for those recorded values. The value of the **relativeTo** attribute is a reference to a time stamp. If it is not given, the time channel values are relative to the beginning timestamps of the individual traces in which they appear.

The following example defines a time channel whose values for a given point are the relative to the timestamp refered to to by `#ts1`:

```
<channel name="T"
            type="integer"
            units="ms"
            respectTo="#ts1"/>
```

If no **<inkSource>** information is provided, or if no value is specified for the **respectTo** attribute, the ink processor cannot make any assumption about the relative timing of points within different traces. Likewise, if no units are specified, no assumption can be made about the units of the time channel data.

3.1.8 User Defined Channels

In addition, user-defined channels are allowed, although their interpretation is not required by conforming ink markup processors.

When specifying a number of related channels, it is recommended to use a common prefix. For example, direction-sensitive stylus force could be named FX, FY, FZ.

3.1.9 Specifying Trace Formats

The following example defines a **<traceFormat>** which reports decimal-valued X and Y coordinates for each point, and intermittent boolean values for the states of two buttons B1 and B2, which have default values of F ("false"):

```
<traceFormat xml:id="xyb1b2">

   <channel name="X" type="decimal">
      <mapping type="identity"/>
   </channel>
   <channel name="Y" type="decimal">
      <mapping type="identity"/>
   </channel>

   <intermittentChannels>
      <channel name="B1" type="boolean" default="F">
         <mapping type="identity"/>
      </channel>
      <channel name="B2" type="boolean" default="F">
         <mapping type="identity"/>
      </channel>
   </intermittentChannels>
</traceFormat>
```

The appearance of a **<traceFormat>** element in an ink markup file both defines the format and installs it as the current format for subsequent traces (except within a **<definitions>** block). The **id** attribute of a **<traceFormat>** allows the format to be reused by multiple contexts (see the Context section). If no **<traceFormat>** is specified, the following default format is assumed:

```
<traceFormat xml:id="DefaultTraceFormat">

   <channel name="X" type="decimal"/>
   <channel name="Y" type="decimal"/>

</traceFormat>
```

Thus, in the simplest case, an InkML file may contain nothing but **<trace>** elements.

3.2 Traces

3.2.1 **<trace>** element

ATTRIBUTES:

xml:id = xsd:ID
The identifier for this trace.
Required: no, *Default:* none

type = "penDown" | "penUp" | "indeterminate"
The type of this trace.
Required: no, *Default:* "penDown"

continuation = "begin" | "end" | "middle"
This attribute indicates whether this trace is a continuation trace, and if it is the case, where this trace is located in the set of continuation traces
Required: no, *Default:* none

priorRef = xsd:anyURI
The URI of the trace this one is a continuation of.
*Required:*if and only if **`continuation`** has values **`end`** or **`middle`**, *Default:* none

contextRef = xsd:anyURI
The context for this trace.
Required: no, *Default:* none

brushRef = xsd:anyURI
The brush for this trace.
Required: no, *Default:* Inherited from context.

duration = xsd:decimal
The duration of this trace, in milliseconds.
Required: no, *Default:* unknown

timeOffset = xsd:decimal
The relative timestamp or time-of-day for the start of this trace, in milliseconds.
Required: no, *Default:* unknown

CONTENTS:

The following grammar defines the syntax of the data that appears within a **`<trace>`** element. It is described in Backus-Naur Form (BNF) using the following notation:

- *: 0 or more
- +: 1 or more
- ?: 0 or 1
- (): grouping
- |: separates alternatives
- double quotes surround literals
- #x precedes hex character codes

The grammar is as follows:

```
trace ::=
     point ("," point)*  ","?

point ::=
     value+

value ::=
     qualifier? "-"? decimal | hex | "T" | "F" | "*" | "?"

decimal ::=
     digit+ ("." digit*)? | "." digit+

hex ::=
     "#" (digit | "A" | "B" | "C" | "D" | "E" | "F")+

qualifier ::=
     "!" | "'" | """

digit ::=
     "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

wsp ::=
     #x20 | #x9 | #xD | #xA
```

Additionally, *wsp* **may** occur anywhere except within a *decimal* or *hex* and **must** occur if required to separate two *value*s. Otherwise the longest token is matched. For example, "3245" requires an internal *wsp* character if it is to be interpreted as two decimal numbers, "32" and "45". On the other hand, "0.923.45" will be interpreted as "0.923" and ".45".

The number of *value* tokens appearing within each point must be at least equal to the number of regular channels and be no more than the number of regular channels plus the number of intermittent channels.

The **<trace>** element is used to record the data captured by the digitizer. It contains a sequence of points encoded according to the specification given by the **<traceFormat>** element.

The **type** attribute of a **<trace>** indicates the pen contact state (either "**penUp**" or "**penDown**") during its recording. A value of "**indeterminate**" is used if the contact-state is neither pen-up nor pen-down, and may be either unknown or variable within the trace. For example, a signature may be captured as a single indeterminate trace containing both the actual writing and the trajectory of the pen between strokes.

If a **continuation** attribute is present, it indicates that the current trace is a continuation trace, i.e. its points are a temporally contiguous continuation of (and thus should be connected to) another trace element. The possible values of the attribute are:

- **begin**: the current trace is the first of the set of continuation traces
- **end**: the current trace is the last of the set of continuation traces
- **middle**: the current trace is a continuation trace, but is neither the first nor the last in the set of traces

If the current trace is a continuation trace but is not the first trace in the set (i.e. the **continuation** attribute has value **middle** or **end**) then a **priorRef** attribute must be present and must contain the URI of the trace of which the current trace is a continuation.

Regular channels may be reported as explicit values, differences, or second differences: Prefix symbols are used to indicate the interpretation of a value: a preceding exclamation point (`!`) indicates an explicit value, a single quote (`'`) indicates a single difference, and a double quote prefix (`"`) indicates a second difference. If there is no prefix, then the channel value is interpreted as explicit, difference, or second difference based on the last prefix for the channel. If there is no last prefix, the value is interpreted as explicit.

A second difference encoding must be preceded by a single difference representation; which, in turn, must be preceded with an explicit encoding.

All traces must begin with an explicit value, not with a first or second difference. This is true of continuation traces as well. This allows the location and velocity state information to be discarded at the end of each trace, simplifying parser design.

Intermittent channels are always encoded explicitly, *i.e.* the qualifiers ' and " are not allowed.

Both regular and intermittent channels may be encoded with the wildcard character "*". This wildcard character means either that the value of the channel remains at the previous channel value (if explicit), or that the channel continues integrating with the previous velocity or acceleration values, as appropriate.

Intermittent channels may be encoded with the wildcard character "?". This means that a value of a channel is not given at that point. It is useful when there are several independent intermittent channels, and they do not always report simultaneously, e.g.

```
<trace> 11 12 9, 21 22 ? T, 31 32, 41 42 5, 51 52 ? F</trace>
```

Booleans are encoded as "T" or "F".

For each point in the trace, regular channel values are reported first in the order given by the **<traceFormat>**. If any intermittent values are reported for the point, they are given next, in the order they are specified in the **<traceFormat>**. Unreported intermittent channels are interpreted as though they were given by the wildcard "*".

Here is an example of a trace of 11 points, using the following traceFormat:

```
<traceFormat>

   <channel name="X" type="decimal"/>
   <channel name="Y" type="decimal"/>

   <intermittentChannels>
      <channel name="B1" type="boolean" default="F"/>
      <channel name="B2" type="boolean" default="F"/>
   </intermittentChannels>
</traceFormat>

<trace id = "id4525abc">
   1125 18432,'23'43,"7"-8,3-5,7 -3,6 2,6 8,3 6 T,2 4*T,3 6,3-6 F F
</trace>
```

The trace is interpreted as follows:

Trace	X	Y	vx	vy	B1	B2	Comments
1125 18432	1125	18432	?	?	F	F	button default values
'23'43	1148	18475	23	43	F	F	velocity values

"7"-8	1178	18510	30	35	F	F	acceleration Values
3-5	1211	18540	33	30	F	F	implicit acceleration no whitespace needed
7 -3	1251	18567	40	27	F	F	optional whitespace
6 2	1297	18596	46	29	F	F	whitespace required
6 8	1349	18633	52	37	F	F	
3 6 T	1404	18676	55	43	T	F	an optional value
2 4*T	1461	18723	57	47	T	T	wildcard
3 6	1521	18776	60	53	T	T	optional keep last
3-6 F F	1584	18823	63	47	F	F	optionals

An ink markup generator might also include additional whitespace formatting for clarity. The following trace specification is identical in meaning to the more compact version shown above:

```
<trace id = "id4525abc">
  1125 18432,
  '23 '43,
  "7 "-8,
  3 -5,
  7 -3,
  6 2,
  6 8,
  3 6 T,
  2 4 * T,
  3 6,
  3 -6 F F
</trace>
```

Note: the trace syntax defined here makes the InkML file sizes (as well as the XML DOM trees) smaller while keeping the benefits of XML. However some applications, for instance those concerned with transmitting InkML documents across the Web, might require even smaller file sizes. It is thus recommended (but not required) that InkML implementations support the gzip standard compression scheme (see [*RFC1952*]).

3.3 Trace Collections

InkML provides mechanisms to gather traces into structured collections via the **<traceGroup>** and **<traceView>** elements. These allow multiple traces or groups to be treated as a flat or structured unit for the purposes of referencing, attaching context information, semantic labelling, or application-specific needs. The **<traceGroup>** element gathers **<trace>** or other **<traceGroup>** elements into a unit. The **<traceView>** element refers to existing **<trace>**, **<traceGroup>** or other **<traceView>** elements to provide alternative views or organization on the ink. For example, a diagramming application may record fixed-length **<trace>** packages, organized as continuations within **<traceGroup>** elements, and use **<traceView>** elements to record the logical structure of the diagram.

3.3.1 **<traceGroup>** element

ATTRIBUTES:

xml:id = xsd:ID

The identifier for this traceGroup.
Required: no, *Default:* none

contextRef = xsd:anyURI
The context associated with this traceGroup.
Required: no, *Default:* none

brushRef = xsd:anyURI
The brush associated with this traceGroup.
Required: no, *Default:* none

CONTENTS:

```
( trace | traceGroup| annotation | annotationXML )*
```

The **<traceGroup>** element is used to group successive traces which share common characteristics, such as the same **<traceFormat>**. The brush and context sections describe other contextual values that can be specified for a **<traceGroup>**. In the following example the two traces enclosed in the **<traceGroup>** share the same brush (see the Brushes section for a description of brushes).

```
<traceGroup brushRef="#penA">
   <trace>...</trace>
   <trace>...</trace>
</traceGroup>
```

The **<traceGroup>** element may be used for various purposes, such as the containment of traces according to their properties at the time of capture. The element may be nested, and it may be used as a generic grouping mechanism, e.g. for the semantic labelling of traces.

Trace groups are the primary mechanism for assigning **<context>** to traces in archival ink markup. For additional details about this usage, see the Archival Applications section.

3.3.2 <traceView> element

ATTRIBUTES:

xml:id = xsd:ID
The identifier for this traceView.
Required: no, *Default:* none

contextRef = xsd:anyURI
The context associated with this traceView.
Required: no, *Default:* none

traceDataRef = xsd:anyURI

A URI reference to a **<trace>**, **<traceGroup>** or **<traceView>** element.
Required: no, *Default:* none

from = xsd:integer [':' xsd:integer]*
The index of the first point that this **<traceView>** element annotates.
Required: no, *Default:* the index of the first referenced point (see prose)

to = xsd:integer[':' xsd:integer]*
The index of the last point in the trace or trace group that this **<traceView>** element annotates.
Required: no, *Default:* the index of the last referenced point (see prose)

CONTENTS:

(traceView |annotation |annotationXML) *

The **<traceView>** element is used to group traces by reference from the current document or other documents. The grouping may be used to provide common contextual values or structure for semantic labelling. Additional context information may be supplied via **<annotation>** or **<annotationXML>** child elements. A given **<traceView>** may have either a **traceDataRef** attribute or **<traceView>** children, but not both.

If a **traceDataRef** attribute is given, then a **to** and/or **from** attribute may be given. Together, **traceDataRef**, **from** and **to** refer to another element and select part of it. An **traceDataRef** attribute may refer to a **<trace>**, a **<traceGroup>** or another **<traceView>**.

A missing **from** attribute is equivalent to selecting the first point in the (recursively) first child of the referenced element. A missing **to** attribute is equivalent to selecting the last point in the (recursively) last child of the referenced element. With these defaults, the **<traceView>** selects the portion of the referenced element from the first point to the last point, inclusive. If neither a **to** nor **from** attribute is given, this implies the entire referenced element is selected.

Any value of a **from** or **to** attribute is a colon-separated list of integers, whose meaning is defined as follows: An empty list of integers selects the entire referenced object (point, **<trace>**, **<traceGroup>** or **<traceView>**). If the list is non-empty, then its first element is taken as a 1-based index into the referenced object, and the remaining list is used to select within the object. It is an error to try to select within a single point.

If the referenced object is a **<traceView>**, then the indexing is relative to the tree selected by the **<traceView>**, not relative to the original object.

If a **<traceGroup>** contains continuation traces, they are counted independently.

If a **contextRef** attribute is given, then it overrides the context of the referenced trace data.

EXAMPLES:

Suppose we have the following ink element:

```
<ink>
   <trace xml:id="L1">911 912, 921 922, 931 932</trace>

   <traceGroup xml:id="L2">
       <trace>111 112, 121 122</trace>
       <traceGroup xml:id="L2-Larry">
           <trace>221 212, 221 222</trace>
           <trace>311 312, 321 322</trace>
       </traceGroup>
       <trace>411 412, 421 422</trace>
       <traceGroup>
           <traceGroup>
               <trace xml:id="L2-Moe">521 512, 521 522</trace>
               <trace>611 612, 621 622</trace>
           </traceGroup>
       </traceGroup>
       <trace>711 712, 721 722</trace>
   </traceGroup>

   <traceView xml:id="L3">
       <traceView traceDataRef="#L1" from="2"/>
       <traceView traceDataRef="#L2" from="2" to="4:1:1"/>
   </traceView>

   <traceView xml:id="L4" traceDataRef="#L3" from="1:2" to="2:1:2:1"/>
</ink>
```

With reference "#L1", the **from** index "2" refers to the point (921, 922). With reference "#L2", the **from** index "2" refers to the **<traceGroup>** with id "L2-Larry", the index "4:1:1" refers to the element with id "L2-Moe", the index "4:1:1:2" refers to the point (521, 522), and the index "4:1:1:2:1" is illegal.

The **<traceView>** with id "L3" selects the following structure

```
<traceGroup>
    <trace>921 922, 931 932</trace>

    <traceGroup>
        <traceGroup>
            <trace>221 212, 221 222</trace>
            <trace>311 312, 321 322</trace>
        </traceGroup>
        <trace>411 412, 421 422</trace>
        <traceGroup>
            <traceGroup>
                <trace>521 512, 521 522</trace>
            </traceGroup>
        </traceGroup>
    </traceGroup>

</traceGroup>
```

and the **<traceView>** with id "L4" selects

```
<traceGroup>
    <trace>931 932</trace>
    <traceGroup>
        <traceGroup>
            <trace>221 212, 221 222</trace>
            <trace>311 312</trace>
        </traceGroup>
    </traceGroup>
</traceGroup>
```

---

4 Contexts

A number of details comprise the context in which ink is written and recorded. Examples include the size of the surface the traces were recorded on, the pen tip used or the accuracy of the pressure measurements. This contextual information needs to be captured by InkML in order to fully characterize the recorded ink data. This section defines markup that provides a way to describe this information, including the `<context>` element which provides a means to associate a defined context with trace data.

The format of trace data -- both in the channels available and their particulars -- may vary from device to device, including from stylus to stylus with the same tablet. Therefore, the `<context>` element may refer to or contain a specific `<traceFormat>` and `<inkSource>` element for the device.

As the ink is generated, there may be various context-dependent attributes associated with the pen. For this, a `<brush>` element may be used to record the attributes of the pen during the capture of the digital ink.

The start times of traces are often given relative to a specified point in time. A context may provide a `<timestamp>` element for this.

For applications that require the sharing of ink, contexts may relate their ink to a shared canvas, given by a `<canvas>` element. The trace format of the ink source is related to the trace format of a shared canvas by means of a `<canvasTransform>` element.

4.1 The `<context>` element

This section describes the `<context>` element and its attributes. The context element both provides access to a useful shared context (canvas) and serves as a convenient agglomeration of contextual attributes. It is used by the `<traceGroup>` and `<traceView>` elements to define the complete shared context of a group of traces or may be referred to as part of a context change in streaming mode. In either mode, individual attributes may be overridden at time of use. Additionally, individual traces may refer to a previously defined context (again optionally overriding its attributes) to describe a context change that persists only for the duration of that trace.

Although the use of the `<context>` element and attributes is strongly encouraged, default interpretations are provided so that they are not required in an ink markup file if all trace data is recorded in the same virtual coordinate system, and its relationship to digitizer coordinates is either not needed or unknown.

ATTRIBUTES:

xml:id = xsd:ID
The unique identifier for this context.
Required: no (yes for archival InkML), *Default:* none

contextRef = xsd:anyURI
A previously defined context upon which this context is to be based.
Required: no, *Default:* none

canvasRef = xsd:anyURI
The URI of a **canvas** element for this context.
Required: no, *Default:* "DefaultCanvas", or inherited from **contextRef**

canvasTransformRef = xsd:anyURI
This is a reference to a mapping from the coordinate system of the trace to the coordinate system of the canvas.
Required: no, *Default:* identity, or inherited from **contextRef**

traceFormatRef = xsd:anyURI
A reference to the traceFormat for this context.
Required: no, *Default:* default trace format, or inherited from **contextRef**

inkSourceRef = xsd:anyURI
A reference to the inkSource for this context.
Required: no, *Default:* default capture device, or inherited from **contextRef**

brushRef = xsd:anyURI
A reference to the brush for this context.
Required: no, *Default:* none, or inherited from **contextRef**

timestampRef = xsd:anyURI
A reference to the timestamp for this context.
Required: no, *Default:* none, or inherited from **contextRef**

CONTENTS:

canvas?
canvasTransform?
traceFormat?
inkSource?
brush?
timestamp?

The **<context>** element consolidates all salient characteristics of one or more ink traces. It may be specified by declaring all non-default attributes, or by referring to a previously defined context and overriding specific attributes. The element is found either in the **definitions** element or as a child of the **ink** element in Streaming InkML

Each constituent part of a context may be provided either by a referencing attribute or as a child element, but not both. Thus it is possible to have either a **traceFormatRef** attribute or a **<traceFormat>** child element, but not both.

4.2 Ink Sources

One of the important requirements for the ink format is to allow accurate recording of metadata about the format and quality of ink as it is reported by the source. The source is typically hardware as embodied in a digitizer device, but may in general be any "virtual" source of ink, such as a software application that is tracking the trajectory of an object. This is accomplished in the **<inkSource>** element, which supports capture of basic information about the make and model of the device and the ink channels captured, as well as very detailed information about a number of source characteristics.

Some of these characteristics are already commonly used in digitizer specifications, while others are somewhat more esoteric, but nonetheless potentially very useful. In

general, these source characteristics describe signal fidelity, allow understanding of the quality of the data, and impose some limits on how the data can be used. They are not intended to be used for repair of bad data from the source.

4.2.1 `<inkSource>` element

ATTRIBUTES:

xml:id = xsd:ID
The unique identifier for this `inkSource` element.
Required: yes

manufacturer = xsd:string
String identifying the digitizer device manufacturer.
Required: no, *Default:* unknown

model = xsd:string
String identifying the digitizer model.
Required: no, *Default:* unknown

serialNo = xsd:string
Unique manufacturer (or other) serial number for the device.
Required: no, *Default:* unknown

specificationRef = xsd:anyURI
URI of a page providing detailed or additional specifications.
Required: no, *Default:* unknown

description = xsd:string
String describing the ink source, especially one implemented in software.
Required: no, *Default:* unknown

CONTENTS:

```
traceFormat
sampleRate?
latency?
activeArea?
srcProperty*
channelProperties?
```

EXAMPLES:

```
<inkSource xml:id = "mytablet"
```

```
      manufacturer = "Example.com"
      model = "ExampleTab 2000 USB"
      specificationRef="http://www.example.com/products/exampletab/2000usb.html">

      <traceFormat href="#XYF"/>

      <sampleRate uniform="True" value="200"/>

      <activeArea size="A6" height="100" width="130" units="mm"/>

      <srcProperty name="weight" value="100" units="g"/>

      <channelProperties>
         <resolution channel="X" value="5000" units="1/in"/>
         <resolution channel="Y" value="5000" units="1/in"/>
         <channelProperty
            channel="Y"
            name="peakRate"
            value="50"
            units="cm/s">
         <resolution channel="F" value="1024" units="dev"/>
      </channelProperties>
   </inkSource>
```

The **<inkSource>** element will allow specification of:

- Manufacturer, model and serial number (of a hardware device)
- Text description of source, and reference (URI) to detailed or additional information
- Trace format - regular and intermitent channels reported by source
- Sampling rate, latency and active area
- Additional properties of the device in the form of name-value-units triples
- Properties of individual channels

The **<inkSource>** block will often be specified by reference to a separate xml document, either local or at some remote URI. Ideally, **<inkSource>** blocks for common devices will become publicly available.

4.2.2 <sampleRate> element

The **<sampleRate>** element captures the rate at which ink samples are reported by the ink source. Many devices report at a uniform rate; other devices may skip duplicate points or report samples only when there is a change in direction. This is indicated using the **uniform** attribute, which must be designated "false" (non-uniform) if **any** pen-down points are skipped or if the sampling is irregular.

ATTRIBUTES:

uniform = xsd:boolean
  Sampling uniformity: Is the sample rate consistent, with no dropped points?
  Required: no, *Default:* unknown
value = xsd:decimal
  The basic sample rate in samples/second.
  Required: yes

CONTENTS:

```
EMPTY
```

EXAMPLES:

```
<sampleRate uniform="True" value="200"/>
```

4.2.3 `<latency>` element

The **`<latency>`** element captures the basic device latency that applies to all channels, in milliseconds, from physical action to the API time stamp. This is specified at the device level, since all channels often are subject to a common processing and communications latency.

ATTRIBUTES:

value = xsd:decimal
Latency in milliseconds.
Required: yes

CONTENTS:

```
EMPTY
```

EXAMPLES:

```
<latency value="50"/>
```

4.2.4 `<activeArea>` element

Many ink capture devices have a notion of active area, which describes the two-dimensional area within which the device is capable of sensing the pen position. This element allows the specification of a rectangular active area.

ATTRIBUTES:

size = xsd:string
The active area, described using an international paper size standard such as

ISO216.
Required: no, *Default:* unknown

height = xsd:decimal
Height of the active area (corresponding to the Y channel).
Required: no, *Default:* unknown

width = xsd:decimal
Width of the active area (corresponding to the X channel).
Required: no, *Default:* unknown

units = xsd:string
Units used for width and height.
*Required:*no, *Default:* unknown

CONTENTS:

```
EMPTY
```

EXAMPLES:

```
<activeArea size="A6" height="100" width="130" units="mm"/>;
```

4.2.5 `<srcProperty>` element

The **`<srcProperty>`** element provides a simple mechanism for the capture of additional **numeric** properties of the ink source as a whole.

ATTRIBUTES:

name = xsd:string
Name of property of device or ink source.
Required: yes

value = xsd:decimal
Value of named property.
Required: yes

units = xsd:string
Units used for value.
*Required:*no, *Default:* unknown

CONTENTS:

```
EMPTY
```

EXAMPLES:

```
<srcProperty name="weight" value="100" units="g"/>
```

4.2.6 `<channelProperties>` element

The **`<channelProperties>`** element is meant for describing properties of specific channels reported by the ink source. Properties such as range and resolution may be specified using corresponding elements. For more esoteric properties (from a lay user's standpoint) the generic channelProperty element may be used.

ATTRIBUTES:

None

CONTENTS:

channelProperty*

EXAMPLES:

```
<channelProperties>
    <channelProperty channel="X" name="resolution" value="5000" units="1/in"/>
    <channelProperty channel="Y" name="resolution" value="5000" units="1/in"/>
    <channelProperty channel="Y" name="peakRate"   value="50"   units="cm/s">
    <channelProperty channel="F" name="resolution" value="1024" units="dev"/>
</channelProperties>
```

4.2.7 `<channelProperty>` element

The **`<channelProperty>`** element provides a simple mechanism for the capture of additional **numeric** properties of specific channels when known and appropriate. The following channel property names, with their specified meanings, are reserved. Other properties may be defined by the user.

Property name	Interpretation
threshold	Threshold - e.g. for a binary channel, the threshold force at which the tip switch is activated

resolution	Resolution - the scale of the values recorded. This may be expressed as fractions of a unit, e.g. 1/1000 *in* (inches), 0.1 *mm*, 1 *deg* (degrees). It may also be expressed, more more popularly, in inverse units, e.g. "1000 points per inch" would be given as 1000 in units *1/in*.
quantization	Quantization - the unit of smallest change in the reported values. If the value is reported as integer, this is assumed to be the same as the resolution. Note that if decimal values are recorded for resolution, the quantization of the data may be smaller than the "resolution".
noise	Noise - the RMS value of noise typically observed on the channel. This is distinct from accuracy! It is an indication of the difference observed in the data from the device when the same path is traced out multiple times (e.g. by a robot).
accuracy	Accuracy - the typical accuracy of the data on the channel (e.g. "0.5 mm", "10 degrees" or "0.1 newton") This is the typical difference between the reported position and the actual position of the pen tip (or tilt ...)
crossCoupling	Cross-coupling - the distortion in the data from one channel due to changes in another channel. For example, the X and Y coordinates in an electromagnetic digitizer are influenced by the tilt of the pen. This would be specified by dX/dOTx = ... or max delta X vs. OTx = ... If the influencing channels are also recorded, and the cross-couplings are accurately specified, it may be possible to compensate for the cross-coupling by subtracting the influence, at the expense of higher noise. The cross-coupling is always expressed in the units of the two channels, e.g. if X mm and OTx is in degrees, then cross-coupling is in mm/deg.
skew	Skew - the temporal skew of this channel relative to the basic device latency, if any. For example, some devices actually sample X and Y at different points in time, so one might have a skew of -5 millisecond, and the other +5 millisecond.
minBandwidth	Minimum bandwidth (in Hz) - the minimum bandwidth of the channel, in Hz (not samples/second), i.e., the frequency of input motion up to which the signal is accurate to within 3dB.
peakRate	Peak rate - the maximum speed at which the device can accurately track motion
distortion	Dynamic distortion, e.g., how velocity affects position accuracy. This is expressed in inverse seconds, e.g. 0.01 mm / mm / s. This kind of distortion is often cross channel, but this specification only allows a generic, channel-specific value.

ATTRIBUTES:

channel = xsd:string
The name of the channel. Must be one among those defined by the ink source's trace format.
Required: yes

name = xsd:string
Name of property of device or ink source.
Required: yes

value = xsd:decimal
Value of named property.
Required: yes

units = xsd:string
Units used for value.
*Required:*no, *Default:* unknown

CONTENTS:

```
EMPTY
```

EXAMPLES:

```
<channelProperty channel="F" name="threshold" value="0.1" units="N"/>
<channelProperty channel="X" name="quantization" value="0.01" units="mm"/>
```

4.3 Brushes

Along with trace data, it is often necessary to record certain attributes of the pen during ink capture. For example, in a notetaking application, it is important to be able to distinguish between traces captured while writing as opposed to those which represent erasures. Because these attributes will often be application specific, this specification does not attempt to enumerate the brush attributes which can be associated with a trace. It also does not provide a language for describing brush attributes, since it is possible to imagine attributes which are described using complex functions parameterized by time, pen-tip force, or other factors. Instead, the specification allows for capturing the fact that a given trace was recorded in a particular brush context, leaving the details of precisely defining specific attributes of that context (such as width and color) to a higher-level, application specific layer.

Depending on the application, brush attributes may change frequently. Accordingly, there should be a concise mechanism to assign the attributes for an individual trace. On the other hand, it is likely that many traces will be recorded using the same sets of attributes; therefore, it should not be necessary to explicitly state the attributes of every trace (again, for reasons of conciseness). Furthermore, it should be possible to define

entities which encompass these attribute sets and refer to them rather than listing the entire set each time. Since many attribute sets will be similar to one another, it should also be possible to inherit attributes from a prior set while overriding some of the attributes in the set.

4.3.1 `<brush>` element

ATTRIBUTES:

xml:id = xsd:ID
The unique identifier for this brush.
Required: yes

brushRef = xsd:anyURI
A brush whose attributes are inherited by this brush.
Required: no, *Default:* none

CONTENTS:

```
(annotation | annotationXML)*
```

In the ink markup, brush attributes are described by the **<brush>** element. This element allows for the definition of reusable sets of brush attributes which may be associated with traces. For reference purposes, a brush specifies an identifier which can be used to refer to the brush. A brush can inherit the attributes of another **<brush>** element by including a brushRef attribute which contains the id of the referenced brush. As noted above, the definitions of specific brush attributes such as color and width are left to the application.

Brush attributes are associated with traces using the brushRef attribute. When it appears as an attribute of an individual **<trace>**, the brushRef specifies the brush attributes for that trace. When it appears as an attribute of a **<traceGroup>** element, the brushRef specifies the common brush attributes for all traces enclosed in the **<traceGroup>**. Within the **<traceGroup>**, an individual trace may still override the traceGroup's brush attributes using a brushRef attribute.

Brush attributes can also be associated with a context by including the brushRef attribute on a **<context>** element. Any traces which reference the context using a contextRef attribute are assigned the brush attributes defined by the context. If a trace includes both brushRef and contextRef attributes, the brushRef overrides any brush attributes given by the contextRef.

In streaming ink markup, brushes are assigned to a trace according to the current brush, which can be set using the **<context>** and **<brush>** elements. See section Streaming Applications for a detailed description of streaming mode.

4.4 Timestamps

Timestamping of traces is supported by the **<timestamp>** element and the **timestampRef**, **timeOffset** and **duration** attributes of the **<trace>** element. For ease of

processing, all timestamps are expressed in milliseconds. Finer-grained timestamps are obtained using fractional values.

4.4.1 `<timestamp>` element

ATTRIBUTES:

xml:id = xsd:ID
The identifier for this timestamp.
Required: yes

time = xsd:decimal
The absolute time for this timestamp, in milliseconds since 1 January 1970 00:00:00 UTC.
Required: no, *Default:* none.

timestampRef = xsd:anyURI
The absolute time for this timestamp, given as a reference to another timestamp.
Required: no, *Default:* none

timeString = xsd:dateTime
The absolute time for this timestamp, given in a human-readable standard format.
Required: no, *Default:* none.

timeOffset = xsd:decimal
The relative time for this reference timestamp, in milliseconds.
Required: No. *Default.* 0

CONTENTS:

```
EMPTY
```

The **`<timestamp>`** element establishes a reference timestamp which can then be used for relative timestamping of traces.

At most one of the attributes **time**, **timestampRef** or **timeString** may be given. The time thus given, plus the value of the attribute **timeOffset**, gives the time value of the timestamp.

If none of **time**, **timestampRef** or **timeString** are given, then the timestamp refers to some unspecified moment in time. This is useful when the timestamp is referenced by multiple elements to provide relative timing information.

The four examples below illustrate the establishment of various reference timestamps. The first **`<timestamp>`** element, ts001, refers to January 2, 2004 at 7:00am, UTC. The second establishes timestamp ts002 which refers to January 2, 2004 at 7:10am, UTC (10 minutes after the refernce timestamp ts001), and the third time stamp, ts003, gives the same time using the **timeString** attribute. The fourth creates ts004 with time January 2, 2004 at 7:10:04.32, UTC (4.32 seconds after the timestamp of trace ts002).

```
<timestamp xml:id="ts001" time="1073026800000"/>
<timestamp xml:id="ts002" timeOffset="600000" timestampRef="#ts001"/>
<timestamp xml:id="ts003" timeString="2004-01-02T07:10:00Z"/>
<timestamp xml:id="ts004" timeOffset="4320" timestampRef="#ts002"/>
```

4.5 The Default Context

Ink traces may specify their contexts explicitly, using a **contextRef** attribute, or implicitly, in which case they use a default context.

Explicitily referenced **<context>** elements may occur in a **<definitions>** element, elsewhere in the same document or in other documents. Explicit contexts are typically used in archival ink applications.

Traces that do not make explicit reference to a context occur in a default context. This is established by the sequence of elements in the **<ink>** element. Initially the default context is empty and uses defaults for all properties, including a default trace format, default canvas, etc. Then, interspersed with ink data, other elements may occur that alter the default context. These elements are **<brush>**, **<context>**, **<traceFormat>**, **<inkSource>** and **<timestamp>**. As the ink is processed from the first child onward, whenever one of these elements is encountered, it is installed as the default to be used by traces. These are used by traces that do not otherwise specify these properties.

The default context may be explicitly specified using the URI "**#DefaultContext**".

---

5 Canvases

InkML provides support for applications that are required to combine ink from multiple sources. This may arise, for example, from real-time collaboration among several devices, from multiple ink annotations on the same base document or multiple pens operating on the same surface. To support these applications, InkML uses the concept of a shared space, called a *canvas*.

A canvas is specified using a **<canvas>** element, and is typically referred to by one or more **<context>** elements. These contexts may each have their own set of ink capture characteristics and trace formats. In order to map traces from a particular context to a canvas, and vice versa, each context provides its own canvas transform, inverse transform or both.

A context neither referencing nor inheriting a canvas uses a default canvas, sufficient to allow simple single-canvas sharing without further action on the part of devices or applications.

Each canvas defines its dimensions by giving a **<traceFormat>** element. Its channel declarations may specify minimum and/or maximum values, an orientation and units. If no minimum or maximum is given for a channel of integer or decimal type, then it is unbounded in that direction.

If a canvas is bounded in any direction, then all traces defined on that canvas must be contained inside its limits. There may be applications where strokes appear outside of the canvas. In these cases the processing of out-of-bounds traces is not defined by the specification.

Although canvases are virtual spaces, each of the coordinates may be assigned a unit

of measure. This allows collaborating parties to establish a common notion of scale.

An example use for such a shared canvas might be a single ink markup stream or file that contains traces captured on a tablet computer, a PDA device, and an opaque graphics tablet attached to a desktop computer. The size of these traces on each ink source and corresponding display might differ, yet it may be necessary to relate these traces to one another. They could represent scribbles on a shared electronic whiteboard, annotations of a common document, or the markings of two players in a distributed tic-tac-toe game.

The trace data for these different ink sessions could be recorded using the same set of virtual coordinates; however, it is often useful, and may even be necessary at times, to record the data in the ink source coordinates, in order to more precisely represent the original capture conditions, for compactness, or to avoid round-off errors that might be associated with the use of a common coordinate system. Thus we define the concept of a "canvas transform", which can vary according to the ink source. The default transform is the identity. It is also possible to specify the mapping from the canvas back to the coordinates of the original trace format. This is useful in collaborative ink applications where ink added to the canvas from one source must be interpreted in the frame of reference of the other sources. It is not always necessary to specify the inverse transform. If the canvas tranform is given as an affine map of full rank, then it may be inverted numerically. Likewise if coordinates are transformed by a lookup table with linear interpolation, then the mapping may be inverted numerically. In all other cases the inverse transformation must be provided if the inverse mapping is required.

5.1 `<canvas>` element

The **`<canvas>`** element provides the virtual coordinate system, which uniquely identifies a shared virtual space for cooperation of ink applications. Together with the trace-to-canvas coordinate transform (discussed below), it provides a common frame of reference for ink collected in multiple sessions on different devices.

ATTRIBUTES:

xml:id = xsd:ID
  The unique identifier for this element.
  Required: no, *Default:* none.

traceFormatRef = xsd:anyURI
  A link to a **`<traceFormat>`** element.
  Required: no, *Default:* none.

CONTENTS:

traceFormat?

A **`<canvas>`** element must have an associated **`<traceFormat>`**, which may either be given as a child element or referred to by a **`traceFormatRef`** attribute. The coordinate

space of the canvas is given by the regular channels of the trace format and any intermittent channels are ignored.

Example:

```
<canvas id="A4PaperCanvas">
  <traceFormat>
      <channel name="X" type="decimal" min="0" max="210" units="mm"/>
      <channel name="Y" type="decimal" min="0" max="297" units="mm"/>
  </traceFormat>
</canvas>
```

5.2 <canvasTransform> element

ATTRIBUTES:

xml:id = xsd:ID
The identifier for this canvas transform.
Required: no, *Default:* none

invertible = xsd:boolean
Required: no, *Default:* false

CONTENTS:

mapping **mapping**?

The **<canvasTransform>** element is used to relate two coordinate systems. The source and target coordinate systems are ultimately defined in terms of **<traceFormat>** elements. These trace formats may either be given directly, or indirectly by **<inkSource>**, **<context>** or other **<canvas>** elements. In general, the source and target coordinate systems may involve a different number and type of coordinates, or have different ranges and orientation for the same dimension.

The contents of the **<canvasTransform>** consists of one or two **<mapping>** elements. If there is only one, then it is the mapping from the source to the target coordinate system, where the meaning of "source" and "target" is determined by the use. If there are two children, the first is the mapping from the source to the target and the second is the inverse mapping from the target back to the source.

The transform and its inverse need not be full inverses in the mathematical sense. If a transform is from a trace format to a canvas with fewer coordinates, then the inverse transform may map from the canvas back to the original trace format by supplying default values for the coordinates not in the canvas. This would occur, for example, if a party were sharing ink from a device with a force channel with a canvas with only spatial coordinates.

For certain classes of mappings, the inverse mapping may be determined automatically.

These are mappings of type "identity", "affine" (for matrices of full rank), "lookup" (univariate, with linear interpolation), and "product" mappings of these. In this case, it is possible to specify that an inverse should be determined automatically by giving only the forward transform and specifying a value of `true` for the **invertible** attribute.

For an application to give only the inverse transform, it should supply the forward transform as an unknown mapping:

```
<canvasTransform>
   <mapping type="unknown"/>
   <mapping mappingRef="#map001"/>
</canvasTransform>
```

5.3 The Default Canvas

The default canvas has two real-valued coordinates X and Y, both unbounded in the positive and negative directions. More precisely, the default canvas is made available as though the following element were included in each InkML document:

```
<canvas xml:id="DefaultCanvas">
  <traceFormat>
    <channel name="X"
          type="decimal" default="0" orientation="+ve" units="em"/>
    <channel name="Y"
          type="decimal" default="0" orientation="+ve" units="em"/>
  </traceFormat>
</canvas>
```

---

6 Generics

This section describes components of the ink markup which are applicable to multiple aspects of the ink markup.

6.1 Mappings

The **<mapping>** element provides a uniform syntax for the various uses of mappings in the ink markup. The element has an **id** attribute, which allows a particular mapping to be applied in multiple places. When a previously defined mapping is reused, the **mappingRef** attribute is used to refer to the **<mapping>** element, which might be defined in a **<definitions>** block. Mappings appear in the following different places in InkML:

1. In a **<channel>** element of a **<traceFormat>**, the **<mapping>** element is used to describe the transformation from the values actually produced by the device to the values recorded in the trace data.
2. In a **<crossCoupling>** element, a mapping can be used to specify the cross-coupling effect of one or multiple channels on another channel.
3. Used by a **<canvasTransform>**, a mapping may be used to specify the forward or inverse transformations between an ink source and a canvas coordinate system.

InkML supports several types of mappings: unknown, identity, lookup table, affine, formula (specified using a subset of MathML) and cross product. The mapping type is indicated by the **type** attribute of a **<mapping>** element. Note: If no mapping appears for a **<channel>**, it defaults to "unknown", which is safer than assuming that 'X' is identical to the device's 'X' since some filtering or modifications could have been applied. Furthermore, one can specify whether the results of a mapping expression are

absolutely or relatively applied to the current data value. This is done by means of the **apply** attribute. For lookup table mappings in particular, one can determine how to interpret intermediate mapping values. This is specified using the **interpolation** attribute.

6.1.1 `<mapping>` element

ATTRIBUTES

xml:id = xsd:ID
The identifier for this mapping.
Required: no, *Default:* none

type = "identity" | "lookup" | "affine" | "mathml" | "product" | "unknown"
The type for the particular mapping.
Required: no, *Default:* unknown

mappingRef = xsd:anyURI
The ID of a mapping which has previously been defined.
Required: no, *Default:* none

CONTENTS

```
(bind* (table | matrix | mathml:math)?) | mapping *
```

THE IDENTITY MAP

If the `type` attribute has value `identity` then the element is empty.

Identity mappings are specified using an empty mapping element:

```
<mapping xml:id="m01" type="identity" />
```

```
<channel name="X" type="decimal" units="point" default="0">
<mapping  type="identity"/>
</channel>
```

They are used, for example, to define a `<traceFormat>` channel that reports the exact data that is recorded by a corresponding device channel, with no filtering or transformation.

LOOKUP TABLES

If the `type` attribute has value `lookup` then the mapping is a unary function specified by a lookup table given as a `<table>` element containing pairs of values separated by

commas.

AFFINE MAPS

If the **`type`** attribute has value **`affine`** then the contents is a **`<matrix>`** element specifying a linear transformation from *n* source values to *m* target values. All of the source and target values must be of the same type, either integer or real (decimal). A matrix containing only the values 0, 1 and -1 may be used to perform arbitrary permutation and reflection of coordinates. If the affine map computes a real number for an integer coordinate, then the value is rounded to the nearest integer.

MATHML MAPPINGS

If the **`type`** attribute has value **`mathml`** then the content is a subset of MathML restricted to the following subset of Content MathML 2.0 elements defining arithmetic on integers, real numbers and boolean values:

- Numbers: `cn`
- Named constants: `exponentiale, pi, true, false`
- Identifiers: `ci`. These must be associated to channels using a **`<bind>`** element.
- Arithmetic: `plus, minus, times, divide, quotient, rem, power, root, min, max, abs, floor, cieling`
- Elementary classical functions: `sin, cos, tan, arcsin, arccos, arctan, exp, ln, log`
- Logic: `and, or, xor, not`
- Relations: `eq, neq, gt, lt, geq, leq`
- Operator application: `apply`

Additionally, an explicit `list` may be used at the top-most level of the MathML expression when the mapping returns multiple values. This is the case in a coordinate transformation.

Example: The following mapping converts from polar to rectangular coordinates.

The arithmetic operators return values whose type depends on the type of the arguments. The logical operators and relations return boolean values. The elementary functions return real values.

```
<mapping type="mathml" m:xmlns="http://www.w3.org/1998/Math/MathML">
    <bind source="R" variable="r"/>
    <bind source="OTh" variable="theta"/>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
       <list>
           <apply>
              <times>
              <ci>r</ci>
              <apply> <cos/> <ci>theta</ci> </apply>
           </apply>
           <apply>
              <times>
              <ci>r</ci>
              <apply> <sin/> <ci>theta</ci> </apply>
           </apply>
```

```
        </list>
      </math>
</mapping>
```

CROSS PRODUCT MAPS

If the **type** attribute has value **product** then the contents is a set of **<mapping>** elements, each giving values for one or more of the coordinates. This allows a multivariate mapping to compute the different coordinate results according to the most convenient means. For example, spatial coordinates may be transformed using an affine map, button states by a lookup table, and color coordinates using formulas.

6.1.2 <bind> element

ATTRIBUTES

source = xsd:string
Specifies source data values and/or channel to be considered in the mapping.
Required: no, *Default:* none

target = xsd:string
Specifies target data values and/or channel to be considered in the mapping.
Required: no, *Default:* none

column = xsd:string
Specifies the assigned column within a lookup table either for source or target channels.
Required: for lookup table bindings, *Default:* none

variable = xsd:string
Specifies the variable within a formula that represents the current source data/channel.
Required: for mathml bindings, *Default:* none

CONTENTS

EMPTY

The **<bind>** element is provided for binding channels to entities (variable names, lookup table columns) within a mapping, and thus it supports the reuse of predefined mappings. For each type of mapping, the relevant bindings can be expressed by the combined usage of the **<bind>** element's attributes, which are **source**, **target**, **column** and **variable**.

For an identity mapping, if the source channel has a different name than the channel

being defined, this can be specified using a **<bind>** element with a **source** attribute. In the following markup, the **<traceFormat>** channel X contains unmanipulated data from the device's devX channel.

```
<channel name="X">

  <mapping type="identity">
     <bind source="devX">
  </mapping>
</channel>
```

Within a mapping formula (**type="mathml"**), the variable names in the formula need to be bound to particular channel names. This is specified using a combination of **source** and **variable** attributes for binding inputs of the formula, and **target** and **variable** for the output of the formula. This is useful if the same mapping formula is to be reused across multiple channels, like X and Y for example.

```
<mapping xml:id="m06" type="mathml">
   <bind type="setvar" target="X" variable="Q" />
   <math mlns=" http://www.w3.org/1998/Math/MathML ">
      <apply>
         <plus/>
         <ci>Q</ci>
         <cn>10</cn>
      </apply>
   </math>
</mapping>
```

The example shown above means that the channel X is referred to by the variable name Q in the mapping expression "Q+10".

For a lookup table, each index column must be bound to the channel that provides the input for the lookup operation. This is done with a **<bind>** element that specifies **source** and **column** attributes. Similarly, each value column must be bound to the channel that receives the output of the lookup. Its **<bind>** element specifies **target** and **column**.

The following example indicates assignments of channels to columns. It means that values for the channels OTx and P are used to look up the value of the cross-coupling for channel X in the table given by the mapping below:

```
<mapping xml:id="m07" type="lookup">
   <bind target="X" column="1"/>
   <bind source="OTx" column="2"/>
   <bind source="P" column="3"/>
   <table apply="relative" interpolate="floor">
      10    45    512,
      9     45    400,
      8     45    372,
      7     45    418,

      10    50    510,
      9     50    403,
      8     50    302,
      7     50    407,

      10    55    512,
      9     55    410,
      8     55    303,
      7     55    405,

      10    60    512,
      9     60    420,
      8     60    355,
      7     60    401,
```

```
   </table>
</mapping>
```

6.1.3 `<table>` element

ATTRIBUTES

xml:id = xsd:ID
The unique identifier for this **`table`** element.
Required: no

apply = "absolute" | "relative"
Specifies whether the mapping values are used from the table/formula, or whether this table/formula needs to be added to the current data value.
Required: no, *Default:* absolute

interpolation = "floor" | "middle" | "ceiling" | "linear" | "cubic"
Specifies the interpolation between discrete mapping values defined by a lookup table.
Required: no, *Default:* "linear"

CONTENTS

```
((xsd:decimal + ,) * xsd:decimal*) |
((xsd:boolean + ,) * xsd:boolean*)
```

The **`<table>`** gives a set of points for a mapping. The points are given as comma-separated rows. Each row must have the same number of entries. The final row may optionally be followed by a comma. Each row in the table represents a value of the function at one point. Which columns represent the argument(s) and which the result(s) is determined by **`<bind>`** elements.

The entries in the table may either be all numerical or all boolean. They may be derived empirically, by measuring properties of a device, calculated to provide efficient approximation to a numerical function, or give an exhaustive enumeration of values of a function over a finite set of values.

Example:

The following example means that X += 10 if 45 ‰¤ E < 50, X += 9 if 50 < E < 55, *etc*.

```
<channelDef name="X"...>
...
<crossCoupling>
<mapping xml:id="m03" type="lookup" apply="relative" lookup="floor">
        <bind target="X"/>
        <bind source="E"/>
        <table>
             45  10,
             50   9,
```

```
                55   8,
                60   7
            </table>
    </mapping>
    </crossCoupling>
    ...
    </channelDef>
```

Tables may have more than two columns, with some of them determining others.

The value of the **interpolation** attribute defines the behavior for indices that don't appear in a numerical table. The following summarizes the behavior of the above table for the various values of **interpolation**:

"floor"	`X += 10  if 45 <= E < 50,`
	`X += 9   if 50 <= E < 55,`
	`...`

"middle"	`X += 10  if E <= 47.5,`
	`X += 9   if 47.5 < E <= 52.5,`
	`...`

"ceiling"	`X += 10  if E <= 45,`
	`X += 9  if 56 < E <= 50,`
	`...`

"linear"	Piece-wise linear interpolation.
"cubic"	Interpolation by cubic splines. This option may be used only for univariate mappings and requires the table have at least 4 points.

The **interpolation** attribute may not be used with boolean tables.

6.1.4 `<matrix>` element

ATTRIBUTES

xml:id = xsd:ID
The unique identifier for this **`matrix`** element.
Required: no

CONTENTS

```
(xsd:decimal + ",")* xsd:decimal *
```

The **<matrix>** element provides the entries for an affine mapping from *n* source values to *m* target values. An affine mapping consists of a linear transformation (multiplication by a matrix) and a shift (adding a vector). The content of the **<matrix>** element is text giving a *m* comma-separated rows of *n+1* numbers each. The final row may optionally be followed by a comma. The first *n* columns specify an *m Ã—n* matrix **M**, and the last column gives a vector **b** of length *m*. If **s** is the source vector of *n* coordinates, then **t = M s + b** is the target vector of *m* coordinates.

6.2 Definitions

6.2.1 <definitions> element

ATTRIBUTES:

none

CONTENTS:

```
( brush | canvas | canvasTransform | context | inkSource | mapping |
timestamp | trace | traceFormat | traceGroup | traceView )*
```

The **<definitions>** element is a container which is used to define reusable content. The definitions within a **<definitions>** block can be referenced by other elements using the appropriate syntax. Content within a **<definitions>** has no impact on the interpretation of traces, unless referenced from outside the **<definitions>**. In order to allow them to be referenced, elements within a **<definitions>** block must include an **id**; attribute. Therefore, an element which is defined inside a **<definitions>** without an **id**, or that is never referenced, serves no purpose.

One of the primary uses of **<definitions>** is to define contextual information. In particular, the elements **<brush>**, **<canvas>**, **<canvasTransform>**, **<context>**, **<inkSource>**, **<mapping>**, **<timestamp>** and **<traceFormat>** may be given inside a **<definitions>**. These may be referenced from other elements by the attributes **brushRef**, **canvasRef**, **canvasTransformRef**, **contextRef**, **inkSourceRef**, **mappingRef**, **timestampRef** and **traceFormatRef**, respectively. Timestamps may also be referenced by the **respectTo** attribute of the **<channel>**element.

Another use of **<definitions>** is to define digital ink traces for later reference. These may be given by **<trace>**, **<traceGroup>** or **<traceView>**, and may be referenced from other elements by the **traceDataRef** attribute. This is useful in archival applications.

The following simple example illustrates usage of the **<definitions>** element.

```
<ink>
   <definitions>
      <brush xml:id="redPen"/>
      <brush xml:id="bluePen"/>
      <traceFormat xml:id="normal"/>
      <traceFormat xml:id="noForce"/>
      <context xml:id="context1"
           brushRef="#redPen"
           traceFormatRef="#normal"/>
      <context xml:id="context2"
           contextRef="#context1"
           brushRef="#bluePen"/>
   </definitions>
   <context contextRef="#context2" traceFormatRef="#noForce"/>
   <context xml:id="context3"/>
</ink>
```

More details on the usage of the **<definitions>** element are provided in the Archival Applications section.

6.3 Annotations

InkML provides generic ways of assigning metadata or semantics to ink via two elements **<annotation>** and **<annotationXML>**, modeled after the corresponding elements in MathML. However since annotations are typically application-specific, InkML does not attempt to prescribe the contents of these elements.

6.3.1 **<annotation>** element

ATTRIBUTES

type = xsd:string
The category of annotation that this element describes, for descriptive purposes only. (Applications may define their own types.)
Required: no
Default: none

encoding = xsd:string
The kind of syntax, standard or convention being used for the values of the annotation, e.g. ISO639 for language codes. *Required:*no
*Default:*none

Other attributes in a namespace other than that of InkML are also allowed, such as general metadata properties (e.g. from the Dublin Core vocabulary) or application-specific attributes.

The **<annotation>** element provides a mechanism for inserting simple textual descriptions in the ink markup. This may be use for multiple purposes. For instance, the text contained in the **<annotation>** may include additional information provided by the user generating InkML, and may be displayed by an InkML consumer rendering a graphical representation of traces. Or it may be used for the indication of metadata such

as the writer, the writing instrument. Another important potential application is the semantic tagging of traces.

EXAMPLE:

```
<ink xmlns:dc="http://dublincore.org/documents/2001/10/26/dcmi-namespace/">
  <annotation type="description">A Sample of Einstein's Writings</annotation>
  <annotation type="writer">Albert Einstein</annotation>
  <annotation type="contentCategory">Text/en</annotation>
  <annotation type="language" encoding="ISO639">en</annotation>
  <annotation dc:language="en"/>

  <trace id="trace1">
    ...
  </trace>
  <traceGroup id="tg1">
    <annotation type="truth">Hello World</annotation>
    <traceGroup>
      <annotation type="truth">Hello</annotation>
      <trace> ... </trace>
      ...
    </traceGroup>
  <traceGroup>
    <annotation type="truth">World</annotation>
      <trace> ... </trace>
        ...
      </traceGroup>
    </traceGroup>

  <traceView href="#tg1">
    <annotation type="style">Cursive</annotation>
  </traceview>
</ink>
```

For semantic tagging, one of the common types of **<annotation>** is "contentCategory", which describes at a basic level the category of content that the traces represent; e.g., "Text/English", "Drawing", "Math", "Music". Such categories are useful for general data identification purposes, and may be essential for selecting data to train handwriting recognizers in different problem domains.

Although largely application-defined, a number of likely, common categories are suggested below.

- Text/<language>[/<script>][/<sub-category>] (e.g., Text/jpn/Kanji, Text/en/SSN)
- Drawing[/<sub-category>] (e.g., Drawing/Sketch, Drawing/Diagram)
- Math
- Music
- Chemistry[<sub-category>]

The language specification may be made using any of the language identifiers specified in ISO 639, using 2-letter codes, 3-letter codes, or country names. Some text may also require a script specification (such as Kanji, Katakana, or Hiragana) in addition to the language.

For some applications it may be useful to provide additional sub-categories defining the type of the data. For example, some suggested sub-categories for Text include:

- SSN (Social Security Number)
- Phone

- Date
- Time
- Currency
- URL

Suggested possible sub-categories for Drawing are:

- Sketch (Not suitable for geometric clean-up)
- Diagram (Suitable for geometric clean-up)

6.3.2 `<annotationXML>` element

ATTRIBUTES

type = xsd:string
The category of annotation that this element describes, for descriptive purposes only. (Applications may define their own types.)
Required: no, *Default:* none

encoding = xsd:string
The kind of syntax, standard or convention being used for the values of the annotation, e.g. ChemML, MathML, rdf, etc.
*Required:*no, *Default:*none

href = xsd:anyURI
A reference to XML content giving giving the annotation. *Required:*no, *Default:*none

> Do we need xlink:href, xlink:type ?

Other attributes in a namespace other than that of InkML are also allowed, such as general metadata properties (e.g. from the Dublin Core vocabulary) or application-specific attributes.

> Should other attributes from other namespaces such as general metadata properties (e.g. from the Dublin Core vocabulary) or application-specific attributes be allowed on annotationXML as well ?

CONTENTS

```
Any XML-based annotation
```

This element allows ink to be annotated with general XML objects. These may be given

either as the content of this element or may be referred to by a **href** attribute, but not both. (If several annotations are desired, several **<annotationXML>** elements should be given.) For instance a handwritten equation may be described using a snippet of MathML, or metadata and semantic annotation may be provided using an XML language.

When annotations of a parent node include the content of the annotations of the child nodes, then one should consider using **<annotationXML>** annotations on the children with **href** attributes referring to sub-trees of the parents annotation in order to maintain linear space complexity in the annotations.

EXAMPLE:

```
<ink>
  <annotation type="description">A Sample of Einstein's Writings</annotation>
  <annotationXML type="metadata" encoding="rdf">
    <rdf:RDF xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:dc = "http://purl.org/dc/elements/1.1/" >
      <rdf:Description about=""
        dc:language="en"
        dc:date="2004-04-11"
        dc:creator="InkML Maker v0.1"
        dc:publisher="Famous Handwritings Ltd."/>
    </rdf:RDF>
  </annotationXML>

  <trace> ... </trace>
  ...
  <trace> ... </trace>
</ink>
```

EXAMPLE:

```
<ink>
  <annotationXML type="truth" encoding="'application/xhtml+xml">
    <html>
      <body>
        <div xml:id="Ch1">
          <h1 xml:id="T1"><span xml:id="W1">Weather</span></h1>
          <p xml:id="P1">
            <span xml:id="W2">The</span> <span xml:id="W3">rain</span>
            ... more words
          </p>
          <p xml:id="P2">...</p>
          ... more paragraphs
          </div>
        </div>
        ... more chapters
      </body>
    </html>
  </annotationXML>

  <traceGroup>
    <annotationXML href="#Ch1"/>
    <traceGroup>
      <annotationXML href="#T1"/>
      <traceGroup>
        <!-- Weather -->
        <annotationXML href="#W1"/>
        <trace>...</trace>
    </traceGroup>
  </traceGroup>
```

```
  <traceGroup>
    <annotationXML href="#P1"/>

    </traceGroup>
      <!-- The -->
      <annotationXML href="#W2"/>
      <trace>...</trace>
    </traceGroup>

    </traceGroup>
      <!-- rain -->
      <annotationXML href="#W3"/>
      <trace>...</trace>
    </traceGroup>
    ... more words in paragraph
  </traceGroup>

  <traceGroup>
    <annotationXML href="#P2"/>
    ... words in paragraph
  </traceGroup>
  ... more paragraphs in chapter.
  </traceGroup>
</ink>
```

If it were not for the sharing of the substructure of the attribute XML data, then each attribute word would be repeated three times (as a word, in a paragraph, and in a chapter), each paragraph would be repeated twice, etc.

6.4 Units

Units are used in several parts of ink mark up. For example channels may report their values with some dimension, such as length, requiring units. Other elements may give values, such as resolution, as quantities in particular units.

The following abbreviations must be recognized as unit attribute values.

Dimension	Unit	Interpretation
length	m	meters
	cm	centimeters
	mm	millimeters
	in	inches
	pt	points (1pt = 1/72 in)
	pc	picas (1pc = 1/22 pt)
	em	ems, the width of a letter "M" in a notional normal size
	ex	exs, the height of a letter "x" in a notional normal size
time	s	seconds
	ms	milliseconds
mass	Kg	kilograms
	g	grams
	mg	milligrams
force	N	Newtons

angle	deg	degrees
	rad	radians
all	%	percentage, relative to *max-min*
	dev	quanta relative to a device resolution. This can correspond to pixels, force levels, clock ticks, etc.

In addition to the units named above, the following expressions must also be recognized:

```
unitExpr ::=
      unit
    | "1"      "/" unit
    | unitExpr "/" unit
    | unitExpr "*" unit

unitPrimitive ::= unit | "(" unitExpr ")"

unit ::= one of the units from the table above,
         with the exception of em, ex, % and dev.
```

Other units are permitted, but need not be recognized by a compliant application.

---

7 Streams and Archives

The ink markup is expected to be utilized in many different scenarios. Ink markup data may be transmitted in substantially real time while exchanging ink messages, or ink documents may be archived for later retrieval or processing.

These examples illustrate two different styles of ink generation and usage. In the former, the markup must facilitate the incremental transmission of a stream of ink data, while in the latter, the markup should provide the structure necessary for operations such as search and interpretation. In order to support both cases, InkML provides archival and streaming modes of usage.

7.1 Archival Applications

In archival usage, contextual elements are defined within a **<definitions>** element and assigned identifiers using the id attribute. References to defined elements are made using the corresponding **brushRef**, **traceFormatRef**, and **contextRef** attributes. The following example:

```
<definitions>
   <brush xml:id="penA"/>
   <brush xml:id="penB"/>

   <traceFormat xml:id="fmt1">

      <channel name="X" type="integer"/>
      <channel name="Y" type="integer"/>
      <channel name="Z" type="integer"/>

   </traceFormat>

   <canvas xml:id="canvasA">
      <traceFormat>
         <channel name="X" type="decimal" min="0" max="200" units="mm">
```

```
            <channel name="Y" type="decimal" min="0" max="150" units="mm">
         </traceFormat>
      </canvas>
      <canvasTransform xml:id="trans1">
         <mapping type="affine">1 0 0 0,0 1 0 0</mapping>
      </canvasTransform>
      <canvasTransform xml:id="trans2">
         <mapping type="affine">2 0 0 0,0 -2 0 0</mapping>
      </canvasTransform>

      <context xml:id="context1"
         canvasRef="#canvasA"
         canvasTransformRef="#trans1"
         traceFormatRef="#fmt1"
         brushRef="#penA"/>

      <context xml:id="context2"
         canvasRef="#canvasA"
         canvasTransformRef="#trans2"
         traceFormatRef="#fmt1"
         brushRef="#penB"/>
   </definitions>
```

defines two brushes ("penA" and "penB"), a traceFormat ("fmt1"), and two contexts ("context1" and "context2") which both refer to the same canvas ("canvasA") and traceFormat ("fmt1"), but with different canvas transforms and brushes. Note the use of the **brushRef**, **traceFormatRef**, **canvasRef** and **canvasTransformRef** attributes to refer to previously defined **<brush>**, **<traceFormat>** **<canvas>** and **<canvasTransform>** elements.

Within the scope of a **<definitions>** element, unspecified attributes of a **<context>** element are assumed to have their default values. This **<definitions>** block:

```
   <definitions>
      <brush xml:id="penA">
      <context xml:id="context1"
         canvasRef="#canvasA"
         brushRef="#penA"/>
   </definitions>
```

defines "context1", which is comprised of "canvasA" with the default canvasTransform and traceFormat (the identity mapping and a traceFormat consisting of decimal X-Y coordinate pairs), and "penA".

A **<context>** element can inherit and override the values of a previously defined context by including a contextRef attribute, so:

```
   <definitions>
      <brush xml:id="#penA"/>
      <context xml:id="context1"
         canvasRef="#canvasA"
         canvasTransformRef="#trans1"/>
      <context xml:id="#context2"
          contextRef="#context1"
          canvasTransformRef="#trans2"
          brushRef="#penA"/>
   </definitions>
```

defines "context2" which shares the same canvas ("canvasA") and traceFormat (the default format) as "context1", but has a different canvasTransform and brush.

Within archival ink markup, traces can either explicitly specify their context through the use of contextRef and brushRef attributes, or they can have their context provided by an enclosing traceGroup. In the following:

```
   <trace xml:id="t001" contextRef="#context1"/>...</trace>
```

```
<trace xml:id="t002" brushRef="#penA"/>...</trace>

<traceGroup contextRef="#context1">
   <trace xml:id="t003">...</trace>
</traceGroup>
```

traces "t001" and "t003" have the context defined by "context1", while trace "t002" has a context consisting of the default canvas, canvasTransform and traceFormat, and "penA".

Traces within a **<traceGroup>** element can also override the context or brush specified by the traceGroup. In this example:

```
<traceGroup contextRef="#context1">
   <trace xml:id="t001">...</trace>
   <trace xml:id="t002" brushRef="#penA">...</trace>
   <trace xml:id="t003">...</trace>
</traceGroup>
```

traces "t001" and "t003" have their context specified by "context1" while trace "t002" overrides the default brush of "context1" with "penA".

A trace or traceGroup can both reference a context and override its brush, as in the following:

```
<trace xml:id="t001" contextRef="#context1" brushRef="#penA">...</trace>
<traceGroup contextRef="#context1" brushRef="#penA">
   <trace xml:id="t002">...</trace>
</traceGroup>
```

which assigns the context specified by "context1" to traces "t001" and "t002", but with "penA" instead of the default brush.

In archival mode, the ink markup processor can straightforwardly determine the context for a given trace by examining only the **<definitions>** blocks within the markup and the enclosing traceGroup for the trace.

7.2 Streaming Applications

In streaming ink markup, changes to trace context are expressed directly using the **<brush>**, **<traceFormat>**, and **<context>** elements. This corresponds to an event-driven model of ink generation, where events which result in contextual changes map directly to elements in the markup.

In the streaming case, the current context consists of the set of canvas, canvasTransform, traceFormat and brush which are associated with subsequent traces in the ink markup. Initially, the current context contains the default canvas, an identity canvasTransform, the default traceFormat, and a brush with no attributes. Each **<brush>**, **<traceFormat>**, and **<context>** element which appears outside of a **<definitions>** element changes the current context accordingly (elements appearing within a **<definitions>** block have no effect on the current context, and behave as described above in the archival section).

The appearance of a **<brush>** element in the ink markup sets the current brush attributes, leaving all other contextual values the same. Likewise, the appearance of a **<traceFormat>** element sets the current traceFormat, and the appearance of a **<context>** element sets the current context.

Outside of a **<definitions>** block, any values which are not specified within a **<context>** element are taken from the current context. For instance, the **<context>**

element in the following example changes the current brush from "penB" to "penA", leaving the canvas, canvasTransform, and traceFormat unchanged from trace "t001" to trace "t002". That is, each context element is taken to inherit from the previously established context.

```
<brush xml:id="penA"/>
<brush xml:id="penB"/>
<trace xml:id="t001">...</trace>
<context brushRef="#penA"/>
<trace xml:id="t002">...</trace>
```

In order to change a contextual value back to its default value, its attribute can be specified with the value "". In the following:

```
<context  canvasRef="#canvasA" brushRef="#penA"/>
<trace xml:id="t001">...</trace>
<context canvasRef="" brushRef=""/>
<trace xml:id="t002">...</trace>
```

Trace "t001" is on "canvasA" and has the brush specified by "penA", while trace "t002" is on the default canvas and has the default brush.

Brushes, traceFormats, and contexts which appear outside of a **<definitions>** block and contain an **id** attribute both set the current context and define contextual elements which can be reused (as shown above for the brushes "penA" and "penB"). This example:

```
<context xml:id="context1"
   canvasRef="#canvasA"
   canvasTransform="#trans1"
   traceFormatRef="#fmt1"
   brushRef="#penA"/>
```

defines a context which can be referred to by its identifier "context1". It also sets the current context to the values specified in the **<context>** element.

A previously defined context is referenced using the **contextRef** attribute of the **<context>** element. For example:

```
<context contextRef="#context1"/>
```

sets the current context to have the values specified by "context1". A **<context>** element can also override values of a previously defined context by including both a **contextRef** attribute and one or more of the **canvasRef**, **canvasTransformRef**, **traceFormatRef** or **brushRef** attributes. The following:

```
<context contextRef="#context1" brushRef="#penB"/>
```

sets the current context to the values specified by "context1", except that the current brush is set to "penB" instead of "penA".

A **<context>** element which inherits and overrides values from a previous context can itself be reused, so the element:

```
<context xml:id="context2" contextRef="#context1" brushRef="#penB"/>
```

defines "context2" which has the same context values as "context1" except for the brush.

Finally, a **<context>** element with only an id has the effect of taking a "snapshot" of the current context which can then be reused. The element:

```
<context xml:id="context3"/>
```

defines "context3", whose values consist of the current canvasRef, canvasTransform, traceFormat, and brush at the point where the element occurs (note that since "context3" does not specify any values, the element has no effect on the current context).

An advantage of the streaming style is that it is easier to express overlapping changes to the individual elements of the context. However, determining the context for a particular trace can require more computation from the ink markup processor, since the entire file may need to be scanned from the beginning in order to establish the current context at the point of the **<trace>** element.

7.3 Archival and Streaming Equivalence

The following examples of archival and streaming ink markup data are equivalent, but they highlight the differences between the two styles:

Archival

```
<ink>
   ...
   <definitions>
      <brush xml:id="penA"/>
      <brush xml:id="penB"/>
      <context xml:id="context1" canvasRef="#canvas1"
         canvasTransform="#trans1" traceFormatRef="format1"/>
      <context xml:id="context2" contextRef="#context1"
         canvasTransform="#trans2"/>
   </definitions>
   <traceGroup contextRef="#context1">
      <trace>...</trace>
      ...
   </traceGroup>
   <traceGroup contextRef="#context2">
      <trace>...</trace>
      ...
   </traceGroup>
   <traceGroup contextRef="#context2" brushRef="#penB">
      <trace>...</trace>
      ...
   </traceGroup>
   <traceGroup contextRef="#context1" brushRef="#penB">
      <trace>...</trace>
      ...
   </traceGroup>
   <traceGroup contextRef="#context1" brushRef="#penA">
      <trace>...</trace>
      ...
   </traceGroup>
</ink>
```

Streaming

```
<ink>
   ...
   <definitions>
      <brush xml:id="penA"/>
      <brush xml:id="penB"/>
   </definitions>
   <context xml:id="context1" canvasRef="#canvas1"
      canvasTransform="#trans1" traceFormatRef="#format1"/>
   <trace>...</trace>
   ...
   <context xml:id="context2" contextRef="#context1"
```

```
        canvasTransform="#trans2"/>
    <trace>...</trace>
    ...
    <context brushRef="#penB"/>
    <trace>...</trace>
    ...
    <context contextRef="#context1"/>
    <trace>...</trace>
    ...
    <context brushRef="#penA"/>
    <trace>...</trace>
    ...
</ink>
```

In the archival case, the context for each trace is simply determined by the **<trace>** element, its enclosing traceGroup, and contextual elements defined in the **<definitions>** block, while in the streaming case, the context for a trace can depend on the entire sequence of context changes up to the point of the **<trace>** element.

However, the streaming case more simply expresses the changes of context involving "penB", "context1", and "penA", whereas the archival case requires the restatement of the unchanged values in the successive traceGroups.

The two styles of ink markup are equally expressive, but impose different requirements on the ink markup processor and generator. Tools to translate from streaming to archival style might also be of use to applications which work on stored ink markup.

A References

[DC]
Dublin Core Metadata Element Set, Version 1.1: Reference Description. http://dublincore.org/documents/dces/ .

[RDF-SYNTAX]
RDF/XML Syntax Specification (Revised), D. Beckett, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/ . Latest version available at http://www.w3.org/TR/rdf-syntax-grammar .

[RFC1952]
GZIP file format specification version 4.3. IETF RFC 1952. http://www.ietf.org /rfc/rfc1952.txt .

[RFC3023]
XML Media Types. IETF RFC 3023. http://www.ietf.org/rfc/rfc3023.txt .

[XMLSCHEMA2]
XML Schema Part 2: Datatypes, A. Malhotra, P. V. Biron, Editors, W3C Recommendation, 2 May 2001, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/ . Latest version available at http://www.w3.org/TR/xmlschema-2/ .

B The InkML Media Type

This appendix registers a new MIME media type, "**application/inkml+xml**".

B.1 Registration of MIME media type application/inkml+xml

MIME media type name:

application

MIME subtype name:

`inkml+xml`

Required parameters:

None.

Optional parameters:

`charset`

This parameter has identical semantics to the `charset` parameter of the `application/xml` media type as specified in [*RFC3023*].

Encoding considerations:

By virtue of InkML content being XML, it has the same considerations when sent as "`application/inkml+xml`" as does XML. See RFC 3023, section 3.2.

Security considerations:

Several InkML instructions may cause arbitrary URIs to be dereferenced. In this case, the security issues of RFC1738, section 6, should be considered.

In addition, because of the extensibility features for InkML, it is possible that "`application/inkml+xml`" may describe content that has security implications beyond those described here. However, if the processor follows only the normative semantics of this specification, this content will be ignored. Only in the case where the processor recognizes and processes the additional content, or where further processing of that content is dispatched to other processors, would security issues potentially arise. And in that case, they would fall outside the domain of this registration document.

Interoperability considerations:

This specification describes processing semantics that dictate behavior that must be followed when dealing with, among other things, unrecognized elements.

Because InkML is extensible, conformant "`application/inkml+xml`" processors can expect that content received is well-formed XML, but it cannot be guaranteed that the content is valid InkML or that the processor will recognize all of the elements and attributes in the document.

Published specification:

This media type registration is for InkML documents as described by this specification.

Additional information:

Magic number(s):

There is no single initial octet sequence that is always present in InkML documents.

File extension(s):

InkML documents are most often identified with the extensions "`.ink`" or "`.inkml`".

Macintosh File Type Code(s):

TEXT

Intended usage:

COMMON

Author/Change controller:

The InkML specification is a work product of the World Wide Web Consortium's Multimodal Interaction Working Group. The W3C has change control over these specifications.

B.2 Fragment Identifiers

For documents labeled as "`application/inkml+xml`", the fragment identifier notation is exactly that for "`application/xml`", as specified in RFC 3023.

C Changes from Previous Working Draft

The following is the list of changes from the previous working draft.

Renamings:

- Renamed the Tx,Ty,A,E,R channels to OTx, OTy, OA, OE, OR to allow the convention that families of channels start with same letter. Otherwise Tx,Ty collide with T.
- `<desc>` has been renamed `<description>`, and `<defs>` has been renamed `<definitions>`.
- Renamed **timeRef** attribute to **timestampRef**.
- The `<captureDevice>` element has been renamed `<inkSource>` to accomodate more general settings.
- The **captureDeviceRef** has been renamed **inkSourceRef**.
- The **canvasTransform** attribute has been renamed **canvasTransformRef**.
- `<traceRef>` is replaced by `<traceView>`, which may be nested. The **href** attribute has been renamed **traceDataRef**.

Simplifications:

- The `<inkSource>` (ex captureDevice) element has been simplified by dropping `<channelDef>` in favor of `<traceFormat>` and `<channelProperties>`.
- The `<regularChannels>` element has been removed. Its contents are now contained directly by the parent `<traceFormat>` element.
- Canvas transforms now use regular mappings. Added `<canvasTransform>` element.
- Simplified use of timestamps. (References to timestamps must be to the timestamp itself, instead of possibly to another element that in turn references a timestamp.) Allow reference to an unspecified arbitrary time.
- Removed **start** attribute from `<trace>`. This duplicated the functionality of **timestampRef** (ex **timeRef**).

- Removed "*" value for **timestampRef** (ex **timeRef**) attribute

Generalizations:

- The content model of `<ink>` has been relaxed.
- The content model of `<definitions>` has been extended to include all definitional items. In particular, `<inkSource>`, `<annotation>`, `<annotationXML>`, are now allowed inside `<definitions>`
- Color and width channels added for optical devices.
- Added **timeString** attribute to `<timestamp>` to recover funtionality from simplification of capture devices.
- Added wildcard character "?" for unknown value of an intermittent channel.
- `<traceGroup>` may be nested
- `<trace>` may contain hex encoded values
- Allowed fractional millisecond values in timestamps and time offsets.
- Canvases have been extended to allow more than X and Y as dimensions.
- The `<mapping>` element has been extended. Mappings now allow affine transformation maps (multiplication by a matrix and adding a vector) and product maps (handling groups of coorinates with separate mappings). Lookup tables are now given by a `<table>` element, rather than textual content, and the `apply` and `interpolation` attributes have been to the table element. Corrected multivariate interpolation example. Table-lookup now allows cubic spline interpolation.
- `<description>` and `<metadata>` have been replaced by a simpler and more general `<annotation>` element for text annotation.
- `<annotationXML>` added.
- `<traceGroup>`, `<traceView>` and `<brush>` may have `<annotation>` or `<annotationXML>` children
- `<definitions>` now allows trace objects as content. This is to define ink that is later refered to by a `<traceView>`, but is not itself yet part of the data stream

Technical changes:

- All `id` attributes are now called `xml:id`.
- All attributes of type `xsd:IDREF` are now `xsd:anyURI`. This allows reference to other documents.
- Trace data points are now separated by commas. This allows parsing of trace data independent of `<traceFormat>`. The colon-semicolon syntax for itermittent points has been dropped.
- Continuation traces are now indicated by the `continuation` attribute on `<trace>`, which indicate the position in the current trace in the set of continuation traces. A new attribute, `priorRef` allows to link a continuation trace to another trace.
- `<bind>` has been moved to inside `<mapping>`.
- `<inkSource>` (ex `<captureDevice>`) has two new attributes **serialNo** and **specificationRef**. The **sampleRate** and **uniform** attributes have been dropped and replaced with a `<sampleRate>` child element.
- `<channel>` has new optional attributes for min/max values, orientation, and units.

Editorial changes:

- Improved the overview.
- Section on timestamps revised.
- The section for `<canvas>` has been revised and made into a top-level chapter.
- Added section on Default Canvas
- Added section on Default Context
- Added section on standard units.
- Section on mappings revised
- Various minor corrections and clarifications.