The Annotation Graph Toolkit: Software Components for Building Linguistic Annotation Tools

Kazuaki Maeda, Steven Bird, Xiaoyi Ma and Haejoong Lee
Linguistic Data Consortium, University of Pennsylvania
3615 Market St., Philadelphia, PA 19104-2608 USA
{maeda, sb, xma, haejoong}@ldc.upenn.edu

ABSTRACT

Annotation graphs provide an efficient and expressive data model for linguistic annotations of time-series data. This paper reports progress on a complete software infrastructure supporting the rapid development of tools for transcribing and annotating time-series data. This general-purpose infrastructure uses annotation graphs as the underlying model, and allows developers to quickly create special-purpose annotation tools using common components. An application programming interface, an I/O library, and graphical user interfaces are described. Our experience has shown us that it is a straightforward task to create new special-purpose annotation tools based on this general-purpose infrastructure.

Keywords

transcription, coding, annotation graph, interlinear text, dialogue annotation

1. INTRODUCTION

Annotation graphs (AGs) provide an efficient and expressive data model for linguistic annotations of time-series data [2]. This paper reports progress on a complete software infrastructure supporting the rapid development of tools for transcribing and annotating time-series data. This general-purpose infrastructure uses annotation graphs as the underlying model, and allows developers to quickly create special-purpose annotation tools using common components. This work is being done in cooperation with the developers of other widely used annotation systems, Transcriber and Emu [1, 3].

The infrastructure is being used in the development of a series of annotation tools at the Linguistic Data Consortium. Several such tools are shown in the paper: one for dialogue annotation, one for telephone conversation transcription, and one for interlinear transcription aligned to speech.

This paper will cover the following points: the application programming interfaces for manipulating annotation graph data and importing data from other formats; the model of inter-component communication which permits easy reuse of software components; and the design of the graphical user interfaces, which have been tailored to be maximally ergonomic for the tasks.

The project homepage is: [`http://www.ldc.upenn.edu/AG/`]. The software tools and software components described in this paper are available through a CVS repository linked from this homepage.

2. ARCHITECTURE

2.1 General Architecture

Existing annotation tools are based on a two level model (Figure 1 Top). The systems we demonstrate are based around a three level model, in which annotation graphs provide a logical level independent of application and physical levels (Figure 1 Bottom). The application level represents special-purpose tools built on top of the general-purpose infrastructure at the logical level.

The system is built from several components which instantiate this model. Figure 2 shows the architecture of the tools currently being developed. Annotation tools, such as the ones discussed below, must provide graphical user interface components for signal visualization and annotation. The communication between components is handled through an extensible event language. An application programming interface for annotation graphs (AG-API) has been developed to support well-formed operations on annotation graphs. This permits applications to abstract away from file format issues, and deal with annotations purely at the logical level.

2.2 The Annotation Graph API

The complete IDL definition of the AG-API is provided in the appendix (also online). Here we describe a few salient features of the API.

The API provides access to internal objects (signals, anchors, annotations etc) using identifiers. Identifiers are strings which contain internal structure. For example, an AG identifier is qualified with an AGSet identifier: `AGSetId:AGId`. Annotations and anchors are doubly qualified: `AGSetId:AGId:AnnotationId`, `AGSetId:AGId:AnchorId`. Thus, it is possible to determine from any given identifiers, its membership in the overall data structure.

The functioning of the API will now be illustrated with a series of examples. Suppose we have already constructed an AG and now wish to create a new anchor. We might have the following API call:

```
CreateAnchor( "agSet12:ag5", 15.234, "sec" );
```

This call would construct a new anchor object and return its identifier: `agSet12:ag5:anchor34`. Alternatively, if we already

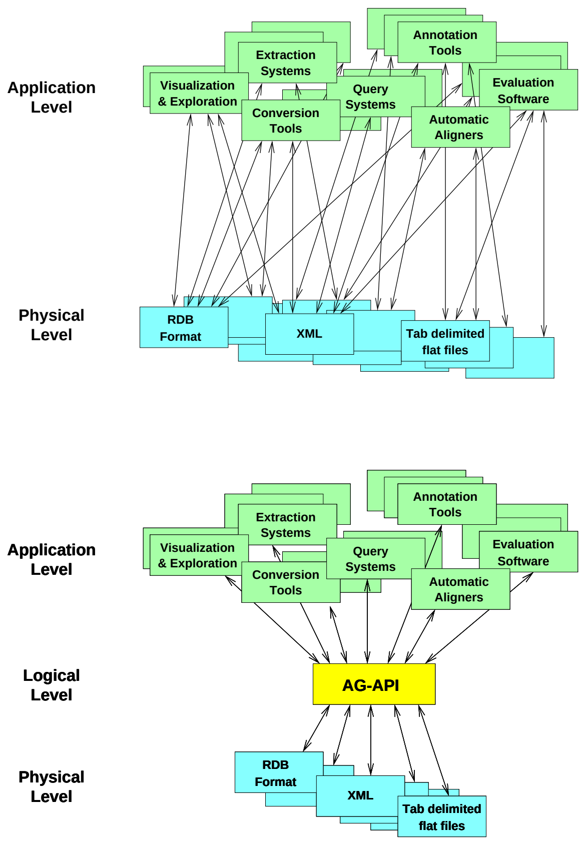


Figure 1: The Two and Three-Level Architectures for Speech Annotation

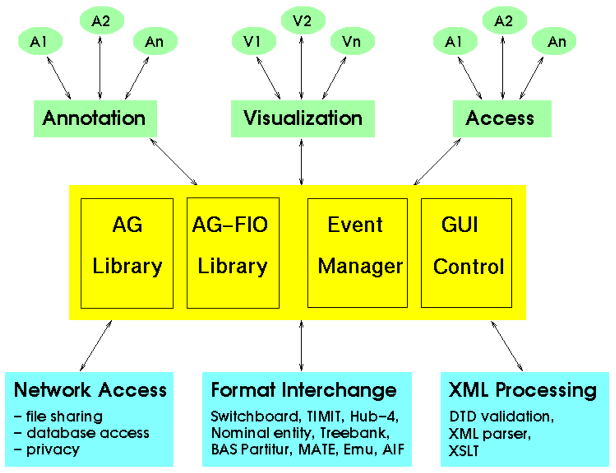


Figure 2: Architecture for Annotation Systems

have an anchor identifier that we wish to use for this new anchor (e.g. because we are reading previously created annotation data from a file and do not wish to assign new identifiers), then we could have the following API call:

```
CreateAnchor( "agset12:ag5:anchor34", 15.234, "sec" );
```

This call will return `agset12:ag5:anchor34`.

Once a pair of anchors have been created it is possible to create an annotation which spans them:

```
CreateAnnotation( "agSet12:ag5",
                  "agSet12:ag5:anchor34",
                  "agSet12:ag5:anchor35",
                  "phonetic" );
```

This call will construct an annotation object and return an identifier for it, e.g. `agSet12:ag5:annotation41`. We can now add features to this annotation:

```
SetFeature( "agSet12:ag5:annotation41",
            "date", "1999-07-02" );
```

The implementation maintains indexes on all the features, and also on the temporal information and graph structure, permitting efficient search using a family of functions such as:

```
GetAnnotationSetByFeature( "agSet12:ag5",
                           "date", "1999-07-02" );
```

2.3 A File I/O Library

A file I/O library (AG-FIO) to support creation and export of AG data has been developed. This will eventually handle all widely used annotation formats. Formats currently supported by the AG-FIO library include the TIMIT, BU, Treebank, AIF (ATLAS Interchange Format), Switchboard and BAS Partitur formats.

2.4 Inter-component Communication

Figure 3 shows the structure of an annotation tool in terms of components and their inter-communications.

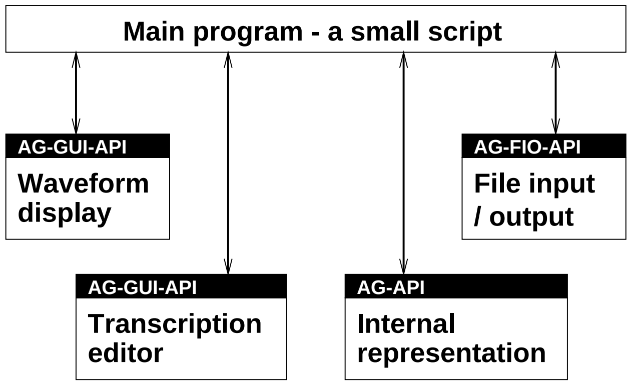


Figure 3: The Structure of an Annotation Tool

The main program is typically a small script which sets up the widgets and provides callback functions to handle widget events. In this example there are four other components which are reused by several annotation tools. The AG and AG-FIO components have already been described. The waveform display component (of which there may be multiple instances) receives instructions to pan and zoom, to play a segment of audio data, and so on. The transcription editor is an annotation component which is specialized for a particular coding task. Most tool customization is accomplished by substituting for this component.

Both GUI components and the main program support a common API for transmitting and receiving events. For example, GUI components have a notion of a “current region” — the timespan which is currently in focus. A waveform component can change an annotation component’s idea of the current region by sending a `SetRegion` event (Figure 4). The same event can also be used in the reverse direction. The main program routes the events between GUI components, calling the AG-API to update the internal representation as needed. With this communication mechanism, it is a straightforward task to add new commands, specific to the annotation task.

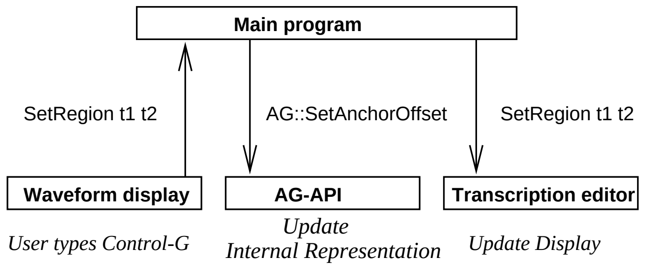


Figure 4: Inter-component Communication

2.5 Reuse of Software Components

The architecture described in this paper allows rapid development of special-purpose annotation tools using common components. In particular, our model of inter-component communication facilitates reuse of software components. The annotation tools described in the next section are not intended for general purpose annotation/transcription tasks; the goal is not to create an “emacs for linguistic annotation”. Instead, they are special-purpose tools based on the general purpose infrastructure. These GUI components can be modified or replaced when building new special-purpose tools.

3. GRAPHICAL USER INTERFACES

3.1 A Spreadsheet Component

The first of the annotation/transcription editor components we describe is a spreadsheet component. In this section, we show two tools that use the spreadsheet component: a dialogue annotation tool and a telephone conversation transcription tool.

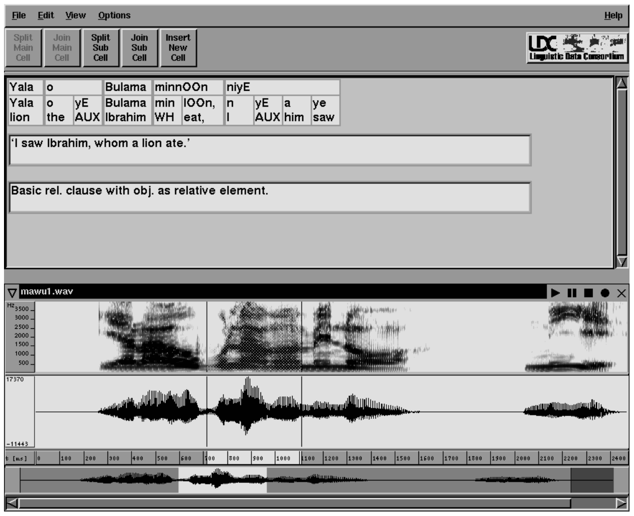
Dialogue annotation consists of assigning a field-structured record to each utterance in each speaker turn. A key challenge is to handle overlapping speaker turns and back-channel cues without disrupting the structure of individual speaker contributions. The tool solves these problems and permits annotations to be aligned to a (multi-channel) recording. The records are displayed in a spreadsheet. Clicking on a row of the spreadsheet causes the corresponding extent of audio signal to be highlighted. As an extended recording is played back, annotated sections are highlighted (both waveform and spreadsheet displays).

Figure 5 shows the tool with a section of the TRAINS/DAMSL corpus [4]. Figure 6 shows another tool designed for transcribing telephone conversations. This latter tool is a version of the dialogue annotation tool, with the columns changed to accommodate the needed fields: in this case, speaker turns and transcriptions. Both of these tools are for two-channel audio files. The audio channel corresponding to the highlighted annotation in the spreadsheet is also highlighted.

3.2 An Interlinear Transcription Component

Interlinear text is a kind of text in which each word is annotated with phonological, morphological and syntactic information (displayed under the word) and each sentence is annotated with a free translation. Our tool permits interlinear transcription aligned to a primary audio signal, for greater accuracy and accountability. Whole words and sub-parts of words can be easily aligned with the audio. Clicking on a piece of the annotation causes the corresponding extent of audio signal to be highlighted. As an extended recording is played back, annotated sections are highlighted (both waveform and interlinear text displays).

The following screenshot shows the tool with some interlinear text from Mawu (a Manding language of the Ivory Coast, West Africa).



Yala	o		Bulama	minnOOn		niyE			
Yala	o	yE	Bulama	min	lOOn,	n	yE	a	ye
lion	the	AUX	Ibrahim	WH	eat,	I	AUX	him	saw

‘I saw Ibrahim, whom a lion ate.’

Basic rel. clause with obj. as relative element.

Figure 7: Interlinear Transcription Tool

3.3 A Waveform Display Component

The tools described above utilize WaveSurfer and Snack developed by Kåre Sjölander and Jonas Beskow [7, 8]. WaveSurfer allows developers to specify event callbacks through a plug-in architecture. We have developed a plug-in for WaveSurfer that enables the inter-component communication described in this paper. In addition to waveforms, it is also possible to show spectrograms and pitch contours of a speech file if the given annotation task requires phonetic analysis of the speech data.

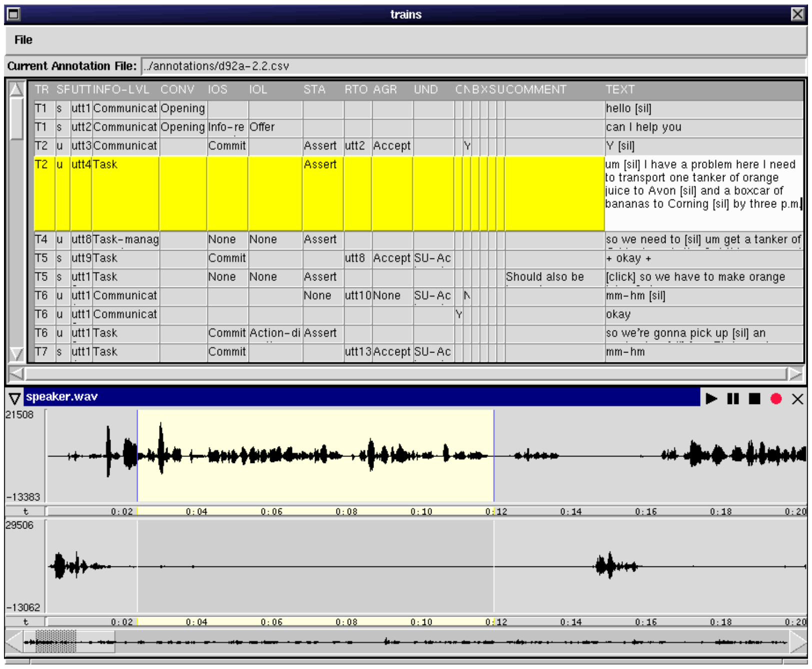
4. FUTURE WORK

4.1 More GUI Components

In addition to the software components discussed in this paper, we plan to develop more components to support various annotation tasks. For example, a video component is being developed, and it will have an associated editor for gestural coding. GUI components for Conversation Analysis (CA) [6] and CHAT [5] are also planned.

4.2 An Annotation Graph Server

We are presently designing a client-side component which presents the same AG-API to the annotation tool, but translates all calls

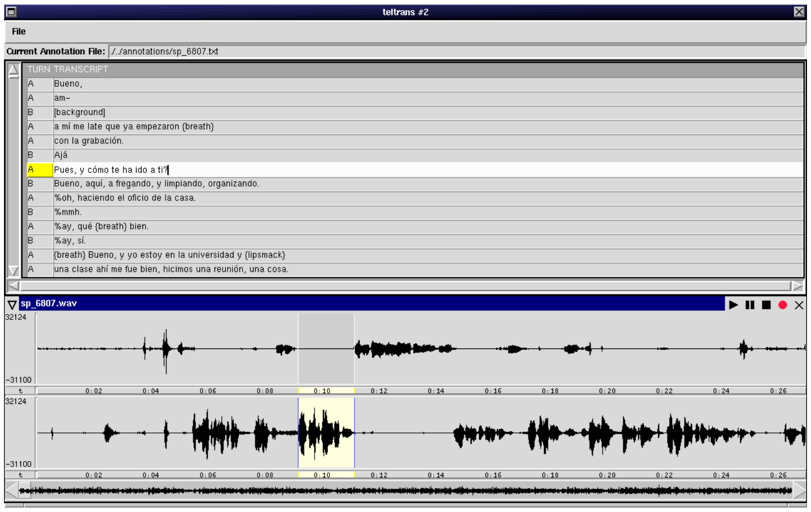

trains

File

Current Annotation File: ../annotations/d92a-2.2.csv

TR	SF	UTT	INFO-LVL	CONV	IOS	IOL	STA	RTO	AGR	UND	C	N	B	X	S	U	COMMENT	TEXT
T1	s	utt1	Communicat	Opening														hello [sil]
T1	s	utt2	Communicat	Opening	Info-re	Offer												can I help you
T2	u	utt3	Communicat		Commit		Assert	utt2	Accept			Y						Y [sil]
T2	u	utt4	Task				Assert											um [sil] I have a problem here I need to transport one tanker of orange juice to Avon [sil] and a boxcar of bananas to Corning [sil] by three p.m.
T4	u	utt8	Task-manag		None	None	Assert											so we need to [sil] um get a tanker of
T5	s	utt9	Task		Commit			utt8	Accept	SU-Ac								+ okay +
T5	s	utt1	Task		None	None	Assert										Should also be	[click] so we have to make orange
T6	u	utt1	Communicat				None	utt10	None	SU-Ac		N						mm-hm [sil]
T6	u	utt1	Communicat								Y							okay
T6	u	utt1	Task		Commit	Action-di	Assert											so we're gonna pick up [sil] an
T7	s	utt1	Task		Commit			utt13	Accept	SU-Ac								mm-hm

Figure 5: Dialogue Annotation Tool for the TRAINS/DAMSL Corpus


teltrans #2

File

Current Annotation File: ./../annotations/sp_6807.txt

TURN	TRANSCRIPT
A	Bueno,
A	am-
B	[background]
A	a mí me late que ya empezaron {breath}
A	con la grabación.
B	Ajá
A	Pues, y cómo te ha ido a ti?
B	Bueno, aquí, a fregando, y limpiando, organizando.
A	%oh, haciendo el oficio de la casa.
B	%mmh.
A	%ay, qué {breath} bien.
B	%ay, sí.
A	{breath} Bueno, y yo estoy en la universidad y {lipsmack}
A	una clase ahí me fue bien, hicimos una reunión, una cosa.

Figure 6: Telephone Conversation Transcription Tool for the CALLFRIEND Spanish Corpus

into SQL and then transmits them to a remote SQL server (see Figure 8). A centralized server could house a potentially large quantity of annotation data, permitting multiple clients to collaboratively construct annotations of shared data. Existing methods for authentication and transaction processing will be be used to ensure the integrity of the data.

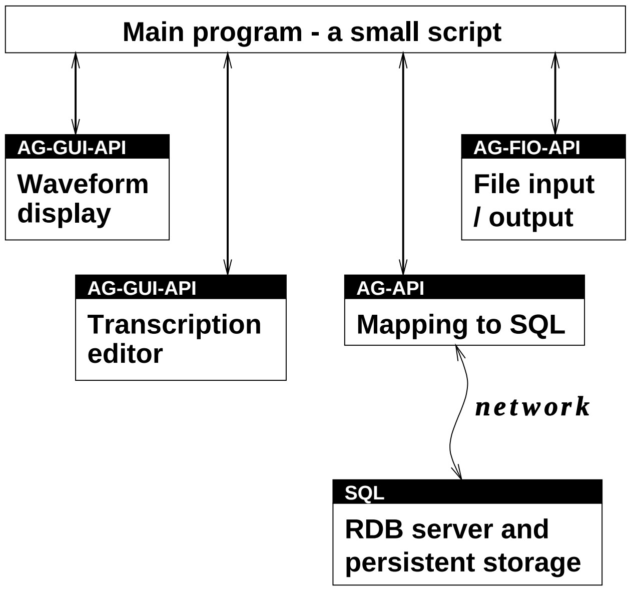


Figure 8: Annotation Tool Connecting to Annotation Server

4.3 Timeline for Development

A general distribution (Version 1.0) of the tools is planned for the early summer, 2001. Additional components and various improvements will be added to future releases. Source code will be available through a source code distribution service, SourceForge ([`http://sourceforge.net/projects/agtk/`]). Further schedule for updates will be posted on our web site: [`http://www.ldc.upenn.edu/AG/`].

5. CONCLUSION

This paper has described a comprehensive infrastructure for developing annotation tools based on annotation graphs. Our experience has shown us that it is a simple matter to construct new special-purpose annotation tools using high-level software components. The tools can be quickly created and deployed, and replaced by new versions as annotation tasks evolve. The components and tools reported here are all being made available under an open source license.

6. ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 9978056 and 9983258.

7. REFERENCES

[1] C. Barras, E. Geoffrois, Z. Wu, and M. Liberman. Transcriber: development and use of a tool for assisting speech corpora production. *Speech Communication*, 33:5–22, 2001.

[2] S. Bird and M. Liberman. A formal framework for linguistic annotation. *Speech Communication*, 33:23–60, 2001.

[3] S. Cassidy and J. Harrington. Multi-level annotation of speech: An overview of the emu speech database management system. *Speech Communication*, 33:61–77, 2001.

[4] D. Jurafsky, E. Shriberg, and D. Biasca. Switchboard SWBD-DAMSL Labeling Project Coder's Manual, Draft 13. Technical Report 97-02, University of Colorado Institute of Cognitive Science, 1997. [`http://stripe.colorado.edu/~jurafsky/manual.august1.html`].

[5] B. MacWhinney. *The CHILDES Project: Tools for Analyzing Talk*. Mahwah, NJ: Lawrence Erlbaum., second edition, 1995. [`http://childes.psy.cmu.edu/`].

[6] E. Schegloff. Reflections on studying prosody in talk-in-interaction. *Language and Speech*, 41:235–60, 1998. [`http://www.sscnet.ucla.edu/soc/faculty/schegloff/prosody/`].

[7] K. Sjölander. The Snack sound toolkit, 2000. [`http://www.speech.kth.se/snack/`].

[8] K. Sjölander and J. Beskow. WaveSurfer – an open source speech tool. In *Proceedings of the 6th International Conference on Spoken Language Processing*, 2000. [`http://www.speech.kth.se/wavesurfer/`].

APPENDIX

A. IDL DEFINITION FOR FLAT AG API

```
interface AG {

typedef string Id;      // generic identifier
typedef string AGSetId; // AGSet identifier
typedef string AGId;    // AG identifier
typedef string AGIds;
   // AG identifiers (space separated list)
typedef string AnnotationId;
   // Annotation identifier
typedef string AnnotationType; // Annotation type
typedef string AnnotationIds;
   // Annotation identifiers (list)
typedef string AnchorId;   // Anchor identifier
typedef string AnchorIds;
   // Anchor identifiers (list)
typedef string TimelineId; // Timeline identifier
typedef string SignalId;   // Signal identifier
typedef string SignalIds;
   // Signal identifiers (list)
typedef string FeatureName;  // feature name
typedef string FeatureNames; // feature name (list)
typedef string FeatureValue; // feature value
typedef string Features;
   // feature=value pairs (list)
typedef string URI;
   // a uniform resource identifier
typedef string MimeClass; // the MIME class
typedef string MimeType; // the MIME type
typedef string Encoding; // the signal encoding
typedef string Unit;     // the unit for offsets
typedef string AnnotationRef;
    // an annotation reference
typedef float  Offset; // the offset into a signal

//// AGSet ////
// Id is AGSetId or AGId
AGId      CreateAG( in Id         id
                    in TimelineId timelineId );
boolean   ExistsAG( in AGId       agId );
void      DeleteAG( in AGId       agId );
AGIds     GetAGIds( in AGSetId    agSetId );

//// Signals ////
TimelineId CreateTimeline( in URI       uri,
                           in MimeClass mimeClass,
                           in MimeType  mimeType,
                           in Encoding  encoding,
                           in Unit      unit,
                           in Track     track  );
TimelineId CreateTimeline( in TimelineId timelineId,
                           in URI        uri,
                           in MimeClass  mimeClass,
                           in MimeType   mimeType,
                           in Encoding   encoding,
                           in Unit       unit,
                           in Track      track);
boolean ExistsTimeline( in TimelineId timelineId );
void DeleteTimeline( in TimelineId timelineId );

// Id may be TimelineId or SignalId
SignalId CreateSignal( in Id        id,
                       in URI       uri,
                       in MimeClass mimeClass,
                       in MimeType  mimeType,
                       in Encoding  encoding,
                       in Unit      unit,
                       in Track     track );
boolean ExistsSignal( in SignalId signalId );
void    DeleteSignal( in SignalId signalId );
SignalIds GetSignals( in TimelineId timelineId );

MimeClass
   GetSignalMimeClass( in SignalId signalId );
MimeType
   GetSignalMimeType( in SignalId signalId );
Encoding GetSignalEncoding( in SignalId signalId );
string GetSignalXlinkType( in SignalId signalId );
string GetSignalXlinkHref( in SignalId signalId );
string  GetSignalUnit(     in SignalId signalId );
Track   GetSignalTrack(    in SignalId signalId );

//// Annotation ////
// Id may be AGId or AnnotationId
AnnotationId CreateAnnotation( in Id id,
             in AnchorId anchorId1,
             in AnchorId anchorId2,
             in AnnotationType annotationType );
boolean ExistsAnnotation
               (in AnnotationId annotationId );
void    DeleteAnnotation
               (in AnnotationId annotationId );
AnnotationId CopyAnnotation
               (in AnnotationId annotationId );
AnnotationIds SplitAnnotation
               (in AnnotationId annotationId );
AnnotationIds NSplitAnnotation(
   in AnnotationId annotationId, in short N );
AnchorId
   GetStartAnchor( in AnnotationId annotationId);
AnchorId GetEndAnchor(
                 in AnnotationId annotationId);
void SetStartAnchor( in AnnotationId annotationId,
                     in AnchorId     anchorId );
void SetEndAnchor( in AnnotationId   annotationId,
                   in AnchorId       anchorId );

Offset
   GetStartOffset( in AnnotationId annotationId );
Offset GetEndOffset(
                 in AnnotationId annotationId );
void SetStartOffset( in AnnotationId annotationId,
                     in Offset       offset );
void   SetEndOffset( in AnnotationId annotationId,
                     in Offset       offset );
// this might be necessary to package up an id
// into a durable reference
AnnotationRef GetRef( in Id id );

//// Features ////
// this is for both the content of an annotation,
// and for the metadata associated with AGSets,
// AGs, Timelines and Signals.
void SetFeature( in Id id,
                 in FeatureName  featureName,
                 in FeatureValue featureValue );
boolean ExistsFeature( in Id id,
                 in FeatureName featureName );
void DeleteFeature( in Id id,
                 in FeatureName featureName );
string GetFeature( in Id id,
                 in FeatureName featureName );
void UnsetFeature( in Id id,
                 in FeatureName featureName );
FeatureNames GetFeatureNames( in Id id );
void         SetFeatures( in Id id,
                 in Features features );
Features     GetFeatures( in Id id );
void         UnsetFeatures( in Id id );

//// Anchor ////
// Id may be AGId or AnchorId
AnchorId CreateAnchor( in Id id,
                       in Offset offset,
                       in Unit   unit,
                       in SignalIds signalIds );
AnchorId CreateAnchor( in Id        id,
                       in SignalIds signalIds );
AnchorId CreateAnchor( in Id        id );
boolean  ExistsAnchor( in AnchorId  anchorId );
void     DeleteAnchor( in AnchorId  anchorId );
void     SetAnchorOffset( in AnchorId anchorId,
                       in Offset offset );
Offset   GetAnchorOffset( in AnchorId anchorId );
void   UnsetAnchorOffset( in AnchorId anchorId );
AnchorId SplitAnchor( in AnchorId anchorId );
AnnotationIds GetIncomingAnnotationSet(
                       in AnchorId anchorId );
AnnotationIds GetOutgoingAnnotationSet(
                       in AnchorId anchorId );
//// Index ////
AnchorIds GetAnchorSet( in AGId agId );
AnchorIds GetAnchorSetByOffset( in AGId agId,
                           in Offset offset,
                           in float  epsilon );
AnchorIds GetAnchorSetNearestOffset(
                           in AGId   agId,
                           in Offset offset );
AnnotationIds
  GetAnnotationSetByFeature( in AGId agId,
                in FeatureName featureName );
AnnotationIds
  GetAnnotationSetByOffset( in AGId agId,
                       in Offset offset );
AnnotationIds
  GetAnnotationSetByType( in AGId agId,
           in AnnotationType annotationType );

//// Ids ////
// Id may be AGId, AnnotationId, AnchorId
AGSetId          GetAGSetId( in Id id );
// Id may be AnnotationId or AnchorId
AGId             GetAGId( in Id id );
// Id may be AGId or SignalId
TimelineId       GetTimelineId( in Id id );
};
```