POWER SCHEMAS WITH STYLUS STUDIO™

January 2004

sonic SOFTWARE™

TABLE OF CONTENTS

> INTRODUCTION

In the XML world, structure is everything. An XML document without a specific, well-defined structure is just an ad-hoc set of tags. Until relatively recently, though, one language to describe those tags, Document Type Definitions or DTDs, lacked the ability to specify information about tags as holders of data. In addition, it was written using a dialect of SGML that runs completely counter to the current XML specification.

It was for these reasons that the XML Schema Definition Language, or XSD specification, was ultimately developed. Utilizing XSD, it's possible to assign to an XML document a separate schema document that describes several things:

> **XML structure,** information about what elements are within other elements, what attributes a given element has, whether an element has text or sub-elements, and so forth.
> **Atomic data type,** indicating the computational type of textual data within an element or attribute (e.g., a string, integer, date, and so forth)
> **Constraints and restrictions,** indicating the range of values a given element or attribute can contain, or the pattern that this data must conform to.
> **Descriptive bindings,** providing a way to associate both text descriptions and thematic meta-data (such as RDF or topic map code) better describing the semantics of an element or attribute.
> **Alternative Sub-Schemas,** making it possible to define sets of possible element groupings (such as the difference between US and British address blocks).

Such schemas can make it much easier to both validate an XML document (check to make sure that its data is at least internally consistent) and to better map XML documents to binary data structures used in traditional programming. However, effective schema design is difficult at the best of times because of the very referential nature of such schemas, and even automation tools such as Sonic's Stylus Studio can only take you so far. However, you *can* use such tools to help you build power schemas that will provide consistency, utility and maintainability for all your XML needs.

> BUILDING SCHEMA INSTANCES

There are two ways to design schemas. The first, and frankly more difficult, is to attempt to create the schema directly element by element. This requires knowing in advance what specific elements already go where, so there is a certain amount of chicken and egg type of problem here — you have to have already designed the schema to design.

The easier solution, and the one discussed in this article, is to create an *instance* of the document, then use schema production tools to generate a schema that is valid for that instance. In this case, the more complete the instance is to your final requirements, the more the tools can do to build the schema from it.

Consider a relatively simple application — defining a schema that can be used by an event calendar. It has to be basic enough to handle a simple internal meeting and yet powerful enough to be useful in giving a schedule for a seminar or rally. This implies a fairly complex "model" (which is essentially what a schema is) that can have options turned off when they aren't needed.

Given that, the first stage is to identify the use cases of your particular schema — examples where it will be used, to more readily identify which information needs to consequently be captured. For the event calendar, the use cases may be as follows:

USE CASE 1. INTERNAL SCHEDULED MEETING

You need to schedule a meeting in your organization to discuss an upcoming seminar the organization is hosting. In this case, the information that needs to be captured includes the name and identity of the person moderating the meeting; the time and duration of the session; the name, description and identity of the meeting; the other participants to the meeting; and where the meeting is being held. Additionally, it may be necessary for participants to confirm their availability via a certain web interface or e-mail address.

USE CASE 2. SEMINAR ANNOUNCEMENT

You wish to advertise the seminar to the general public. In this case, you may have more than one speaker, likely a host organization, a schedule that may be broken into several specific days, a need to sign up ahead of time, and a way of indicating ticket prices.

USE CASE 3. SEMINAR SCHEDULE

You are putting together the track schedules for the seminar. Here, each session needs to be treated as an individual event, while overall seminar information becomes less (or completely not) important. The speaker for each session, a synopsis of the session, links to resources, and track information gets stressed.

When presented like this, the first thing that you may notice is that an "event" describes a fairly broad category of things. Once the use cases are articulated, there may be scenarios that add too much complexity to the requirements. This can be an indication that there are in fact two or more schemas that describe similar concepts. To identify whether that is the case (and to start the actual design) it's worth finding those aspects that are common from one use case to the next.

With the three cases described here, the "core" event contains a title, a general description, a time, a duration, and a location. You could create a first-pass XML document to describe this as follows:

Listing 1. A Simple XML event instance

```
<event>
     <title>XML Seminar Planning Sessions</title>
     <abstract>This meeting will cover the upcoming seminar</abstract>
     <dateTime>January 20, 2003 at 7:00 PM</dateTime>
     <duration>2 hours</duration>
     <address>Ranier (Room 25), Building C</address>
</event>
```

However, there are a number of problems with a simple schema like this. The first is the fact that it is usually necessary for the computer to be able to differentiate between multiple events — two events may have the same title but be completely unrelated.

A second problem comes if a meeting is scheduled to start at a certain time, go for a certain duration, break for lunch, then start again at a later time. The `<dateTime>` and `<duration>` elements actually make up a single unit — call it a schedule element — that may be repeated multiple times within the event.

Similarly, what happens when the address of the meeting is outside of the same building? The structure needs to be more granular with this information — indicating which building is referenced, or even a total mailing address for people who are coming into the meeting from out of town.

The final problem with the above outline deals with the way that times and durations are specified. The values for `<dateTime>` and `<duration>` are useful for humans to read, but require some serious parsing in order to be understood by a computer. By conforming to a distinct standard that XML languages are capable of understanding, you can minimize the amount of processing necessary in consumers of this XML.

A revised `<event>` item, taking into account these factors, may look more like the following:

Listing 2. Getting a little more complex

```
<event id="event-schema-00011521">
     <title>XML Seminar Planning Sessions</title>
     <abstract>This meeting will cover the upcoming seminar</abstract>
     <scheduledTime>
          <dateTime>2003-01-20T19:00:00</dateTime>
          <duration>PT2H</duration>
     </scheduledTime>
     <address>
          <room>Ranier (Room 25)</room>
          <building>C</building>
     </address>
</event>
```

The `id` attribute on the event contains a unique string that identifies the event; the degree of that uniqueness will depend upon the scope of the event — if the events being discussed are local to a group, then uniqueness could be something as simple as keeping track of how many meetings have been held. On the other hand, if the event was part of a distributed hierarchy sent out on the Internet as part of an RSS feed, (as just one example) the id would have to be very unique, and would probably end up relying on something like a UUID (Java) or GUID (Microsoft) to be able to identify the event absolutely.

The `<dateTime>` element may look a little odd if you're not that familiar with XML schema, but it is actually a very convenient and compact notation for representing time. By giving the time in year, month, date, hour, minute and second order (assuming two digits for all but the year) you have a fixed-length string that can be queried ***to retrieve*** specific values in a uniform manner. Moreover, if you order the dates alphanumerically, they will also be in chronological order.

Similarly, the duration element "PT3H" gives a period ("P") of 2 hours ("H"). Periods are given in descending value of time. Thus a period of two days, six hours and eleven minutes is rendered as "P2DT6H11N". Again, while not immediately legible, this format is easy to parse. The T indicates that the values following are granular units of time less than one day.

This schema can in turn be expanded a little bit more. For instance, most events incorporate one or more speakers, each of whom has his own contact information (such as a web page). Moreover, it's perfectly within the realm of possibility to see such a schema being used to describe the schedule of a seminar or other program, with one event being associated with each session in the seminar. This would in turn imply that more than one event could be contained within an events block. Finally, it's entirely possible that each event may be associated with some sort of track or timeline (such as a management vs. technical track).

Listing 3. A Databook.xml instance

```
<datebook id="datebook-schema-00011520"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="c:\Publishing\Sonic\datebook1.xsd">
     <schedule id="datebook-schema-00011521">
   <track>Technical</track>
   <description>This describes the technical track of the
seminar.</description>
   <event id="datebook-schema-00011522">
     <title>Introducing XML</title>
     <description>This is the introductory session for the
seminar.</description>
     <speaker>
         <name>Kurt Cagle</name>
           <email>kurt@kurtcagle.net</email>
           <description>Kurt Cagle is an author specializing in XML
technologies</description>
           <link>http://www.metaphoricalweb.com</link>
     </speaker>
     <scheduledTime>
     <dateTime>2003-01-20T09:00:00</dateTime>
           <duration>PT3H</duration>
     </scheduledTime>
     <address>
<room>Ranier (Room 25)</room>
<building>C</building>
</address>
   </event>
   <event id="datebook-schema-00011523">
      <title>Lunch</title>
      <description>This is a working lunch.</description>
      <scheduledTime>
      <dateTime>2003-01-20T12:00:00</dateTime>
           <duration>PT1H</duration>
      </scheduledTime>
      <address>
<room>Main Ballroom</room>
<building>C</building>
</address>
   </event>
   <event id="datebook-schema-00011524">
```

```
       <title>Working with XSLT</title>
       <description>This session deals with XSLT</description>
       <speaker>
             <name>Ellison Dean</name>
              <email>Ellison@sonic.com</email>
              <description>Ellison is an author specializing in XSLT
technologies</description>
              <link>http://www.stylusstudio.com</link>
       </speaker>
       <scheduledTime>
       <dateTime>2003-01-20T13:00:00</dateTime>
             <duration>PT3H</duration>
       </scheduledTime>
       <address>
<room>Ranier (Room 25)</room>
<building>C</building>
</address>
   </event>
   </schedule>
</datebook>
```

The most obvious change here is the introduction of both the `<datebook>` and `<schedule>` elements. A `<schedule>` is a collection of events, with each event having some immediate topical connection and in general with the events not overlapping one another. For instance, a schedule may contain all of the events of a technical track, while another schedule may contain all of the events of a management track. You can have two events that overlap if they are in different schedules.

A `<datebook>` in turn can be thought of as a collection of schedules, which may or may not be related to one another. Such a date book may actually contain dozens of different schedules, perhaps for transport as part of a web service for updating a calendar or similar interface.

There are other subtleties in this example; an event could have more than one speaker, or none at all (for instance, the lunch event in listing 3). Similarly, an event could have more than one `<scheduledTime>` — for instance, it may start at 9 AM, go to 12PM, break for lunch, restart at 1 PM, go to 5 PM and maybe even spill over into the next day at 9 AM. The scope of the event is important here — if the purpose is to just give the general time listing for the event, even if broken up, then this information would be contained as a set of individual scheduled times. On the other hand, if you have a seminar that has a number of separate events with different speakers and topics within the overall seminar, then it's better to break things into distinct events.

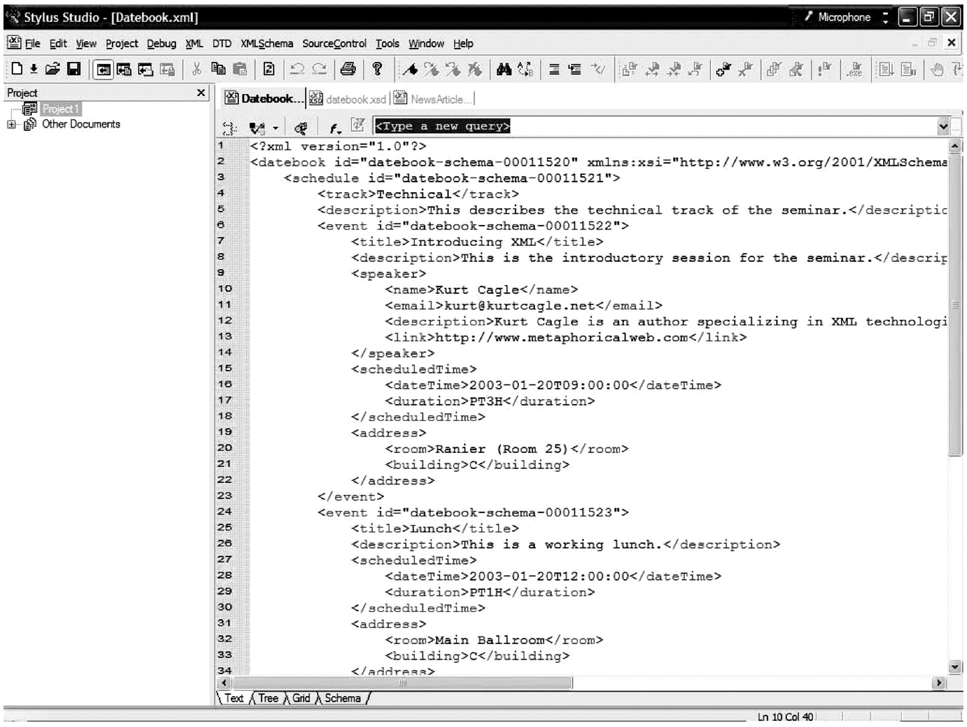
While this is a pretty good design for what a date book schema should produce, it is definitely worth putting as much effort as you can into this initial process of generating use cases and refining the schema through repeated iterative cycles. To overview, the best way of building such schema instances is to work through the following process:

1. Identify those use cases that best describe how the XML schema will be used.
2. Use an iterative process, adding elements that satisfy the larger cases then removing elements that are not strictly germane to the schema.
3. Use element names that are descriptive but not overly long.
4. Use attributes principally for those characteristics that either identify a given element uniquely or that provide links to external resources.
5. Your schema should balance depth with breadth. A structure that is too deep (especially with repeated elements) becomes costly to navigate, while a structure that is too broad (i.e., flat) loses structure.

These principles are good for creating the instance. Generating the schema itself is the next step in the process, and in general is best done with the aid of a tool such as Stylus Studio.

> GENERATING SCHEMAS

Generating a schema from an XML document in Stylus Studio is a straightforward operation. The XML Schema Stylus Studio produces is valid and accurate, and thus well suited to the development of formal schemas that can be used as the basis for enterprise-wide application development.

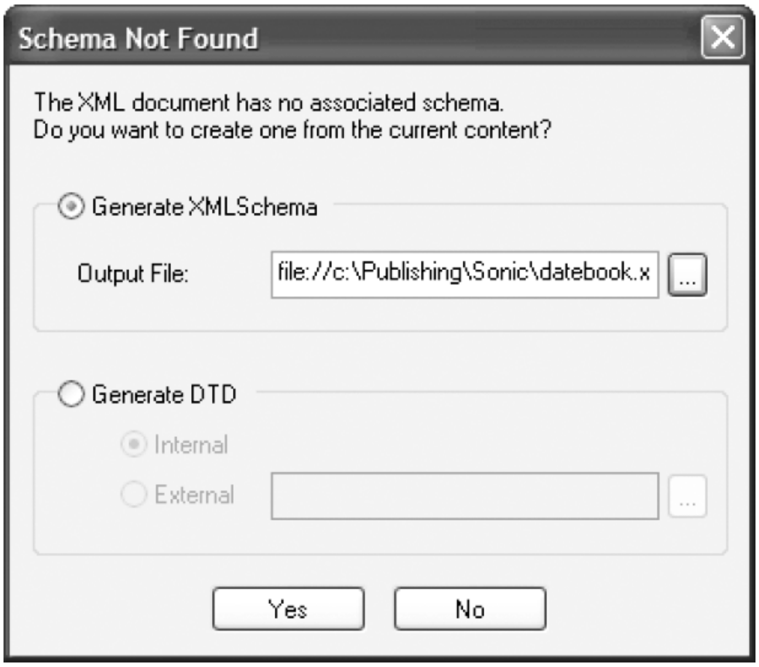


```
<?xml version="1.0"?>
<datebook id="datebook-schema-00011520" xmlns:xsi="http://www.w3.org/2001/XMLSchema
    <schedule id="datebook-schema-00011521">
        <track>Technical</track>
        <description>This describes the technical track of the seminar.</descriptic
        <event id="datebook-schema-00011522">
            <title>Introducing XML</title>
            <description>This is the introductory session for the seminar.</descrip
            <speaker>
                <name>Kurt Cagle</name>
                <email>kurt@kurtcagle.net</email>
                <description>Kurt Cagle is an author specializing in XML technologi
                <link>http://www.metaphoricalweb.com</link>
            </speaker>
            <scheduledTime>
                <dateTime>2003-01-20T09:00:00</dateTime>
                <duration>PT3H</duration>
            </scheduledTime>
            <address>
                <room>Ranier (Room 25)</room>
                <building>C</building>
            </address>
        </event>
        <event id="datebook-schema-00011523">
            <title>Lunch</title>
            <description>This is a working lunch.</description>
            <scheduledTime>
                <dateTime>2003-01-20T12:00:00</dateTime>
                <duration>PT1H</duration>
            </scheduledTime>
            <address>
                <room>Main Ballroom</room>
                <building>C</building>
            </address>
```

Figure 1. The **Datebook.xml** *instance, as shown in Text mode*

Once you have designed your comprehensive instance, the one that handles the most general use case, you're ready to generate the schema itself. To do this, load the document into Stylus Studio as an XML document. At the very bottom of the XML document pane are four tabs: Text, Tree, Grid, and Schema (see Figure 1).

To create a new schema, click on the Schema tab. Because no formal schema has been associated with the XML, Stylus Studio asks if you want to generate a new schema, as shown in Figure 2.



Schema Not Found

The XML document has no associated schema.
Do you want to create one from the current content?

(●) Generate XMLSchema

Output File: file://c:\Publishing\Sonic\datebook.x

( ) Generate DTD

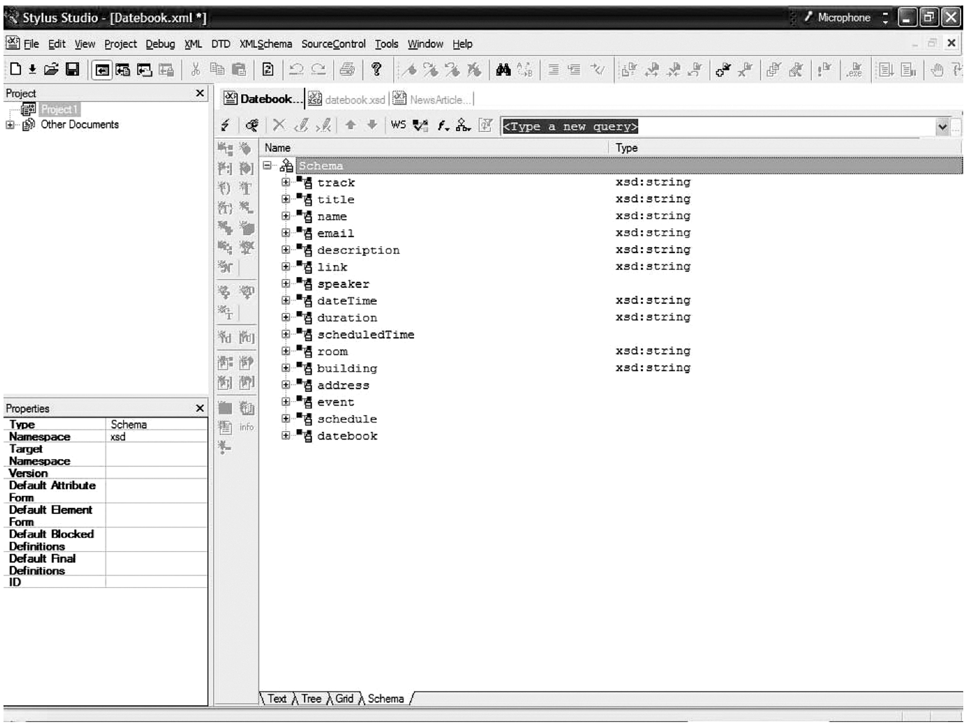
(●) Internal

( ) External

Figure 2. Generating a schema from an existing document

You can create either an XML Schema (an XSD document), or a Document Type Definition (a DTD) from this dialog. Given the fairly strong data-centric nature of this particular document (and the need to define specific data types) you should select Generate XMLSchema, then provide the name and location of the file you want to build. The extension for this file should be .xsd. The example provided here was saved as datebook.xsd.

Once the schema is created, it is displayed as a read-only version within the Schema tab of the Stylus Studio editor. You can review elements, see their relationships, and even perform XPath queries on the result as shown in Figure 3.



Type	Schema
Namespace	xsd
Target Namespace	
Version	
Default Attribute Form	
Default Element Form	
Default Blocked Definitions	
Default Final Definitions	
ID	

Name	Type
Schema	
track	xsd:string
title	xsd:string
name	xsd:string
email	xsd:string
description	xsd:string
link	xsd:string
speaker	
dateTime	xsd:string
duration	xsd:string
scheduledTime	
room	xsd:string
building	xsd:string
address	
event	
schedule	
datebook	

Figure 3. The Read-Only Schema pane of the XML document

Returning to the Text tab, you will notice that one new namespace declaration and one attribute have been added to the `<datebook>` element:

```
<datebook id=”urn:datebook-schema-00011520”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
xsi:noNamespaceSchemaLocation=”file://c:\Publishing\Sonic\datebook.xsd”>
```

The first namespace, associated with the prefix `xsi:`, provides the schema necessary to define core data types (such as floats, integers, dates, and so forth) within the document. The principle reason that this is added here is that this schema also supports another attribute: `xsi:noNamespaceSchemaLocation`. This attribute is used to point to a schema file when no specific namespace association has been made. The value of this attribute is the URL of the schema file, and is used by Stylus Studio to link the schema to the file.

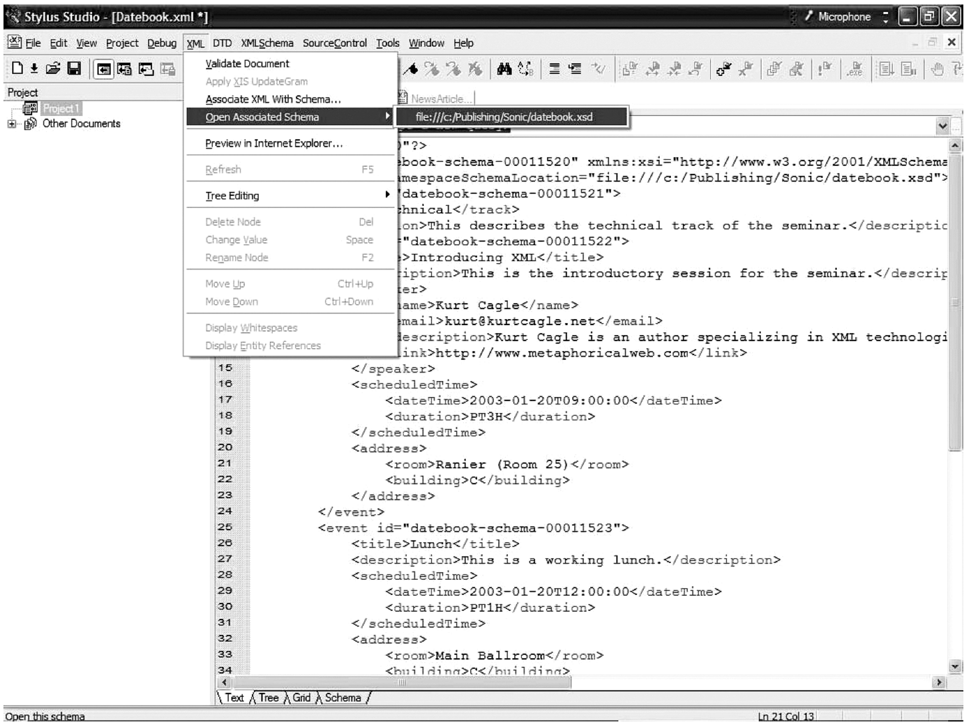
Suppose that you did have a namespace associated with the above instance. For instance, suppose that the datebook instance was associated with the namespace http://www.stylusstudio.com/ns/datebook. To see how this changes things, remove the `xmlns:xsi` namespace declaration and the `xsi:noNamespaceSchemaLocation` attribute, so that the `<datebook>` element is back to the way that it was. Then assign to this element the mentioned namespace as a default namespace, as follows:

```
<datebook id=”urn:datebook-schema-00011520”
xmlns=”http://www.stylusstudio.com/ns/datebook”>...
```

If you repeat the process from above, overwriting the old schema with the new one, the altered <datebook> element will now look like:

```
<datebook id=”urn:datebook-schema-00011520”
xmlns=”http://www.stylusstudio.com/ns/datebook”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
xsi:SchemaLocation=”http://www.stylusstudio.com/ns/datebook
file:///c:/Publishing/Sonic/datebook.xsd”>
```

So how to get at the schema? If you return to the Text view (click the Text tab at the bottom of the editor), you can display the schema by opening the file from the menu, as shown in Figure 4.



```
)"?>
book-schema-00011520" xmlns:xsi="http://www.w3.org/2001/XMLSchema
mespaceSchemaLocation="file:///c:/Publishing/Sonic/datebook.xsd">
'datebook-schema-00011521">
hnical</track>
on>This describes the technical track of the seminar.</descriptic
"datebook-schema-00011522">
>Introducing XML</title>
iption>This is the introductory session for the seminar.</descrip
er>
ame>Kurt Cagle</name>
mail>kurt@kurtcagle.net</email>
escription>Kurt Cagle is an author specializing in XML technologi
ink>http://www.metaphoricalweb.com</link>
          </speaker>
          <scheduledTime>
              <dateTime>2003-01-20T09:00:00</dateTime>
              <duration>PT3H</duration>
          </scheduledTime>
          <address>
              <room>Ranier (Room 25)</room>
              <building>C</building>
          </address>
      </event>
      <event id="datebook-schema-00011523">
          <title>Lunch</title>
          <description>This is a working lunch.</description>
          <scheduledTime>
              <dateTime>2003-01-20T12:00:00</dateTime>
              <duration>PT1H</duration>
          </scheduledTime>
          <address>
              <room>Main Ballroom</room>
              <building>C</building>
```

Figure 4.* Select Open Associated *Schema from the XML menu

The schema that’s generated for the date book with no explicit namespace is given in Listing 4. Pressing Ctrl-Tab lets you switch between the schema and instance view.

Listing 4. Datebook.xsd

```
<xsd:schema xmlns:xsd=’http://www.w3.org/2001/XMLSchema’ >

 <xsd:element name=’track’ type=’xsd:string’>
 </xsd:element>

 <xsd:element name=’title’ type=’xsd:string’>
 </xsd:element>
```

```
<xsd:element name='name' type='xsd:string'>
</xsd:element>

<xsd:element name='email' type='xsd:string'>
</xsd:element>

<xsd:element name='description' type='xsd:string'>
</xsd:element>

<xsd:element name='link' type='xsd:string'>
</xsd:element>

<xsd:element name='speaker'>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref='name'/>
   <xsd:element ref='email'/>
   <xsd:element ref='description'/>
   <xsd:element ref='link'/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:element name='dateTime' type='xsd:string'>
</xsd:element>

<xsd:element name='duration' type='xsd:string'>
</xsd:element>

<xsd:element name='scheduledTime'>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref='dateTime'/>
   <xsd:element ref='duration'/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:element name='room' type='xsd:string'>
</xsd:element>
```

```
<xsd:element name='building' type='xsd:string'>
</xsd:element>

<xsd:element name='address'>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref='room'/>
   <xsd:element ref='building'/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:element name='event'>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref='title'/>
   <xsd:element ref='description'/>
   <xsd:sequence minOccurs='0' maxOccurs='1'>
    <xsd:element ref='speaker'/>
   </xsd:sequence>
   <xsd:element ref='scheduledTime'/>
   <xsd:element ref='address'/>
  </xsd:sequence>
  <xsd:attribute name='id' type='xsd:ID' use='required'/>
 </xsd:complexType>
</xsd:element>

<xsd:element name='schedule'>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref='track'/>
   <xsd:element ref='description'/>
   <xsd:sequence maxOccurs='unbounded'>
    <xsd:element ref='event'/>
   </xsd:sequence>
  </xsd:sequence>
  <xsd:attribute name='id' type='xsd:ID' use='required'/>
 </xsd:complexType>
</xsd:element>
```

```
 <xsd:element name='datebook'>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref='schedule'/>
   </xsd:sequence>
   <xsd:attribute name='id' type='xsd:ID' use='required'/>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

In general, there is no preferred order within a schema document — you generally have to infer the "containing" element of a schema based upon which `<xsd:element>` references the highest elements in the tree. In practice, Stylus Studio and other schema generators usually place the document element (the element in an XML tree that everything else descends from) as the last element in the schema. Thus, the document element `<datebook>` is defined as follows:

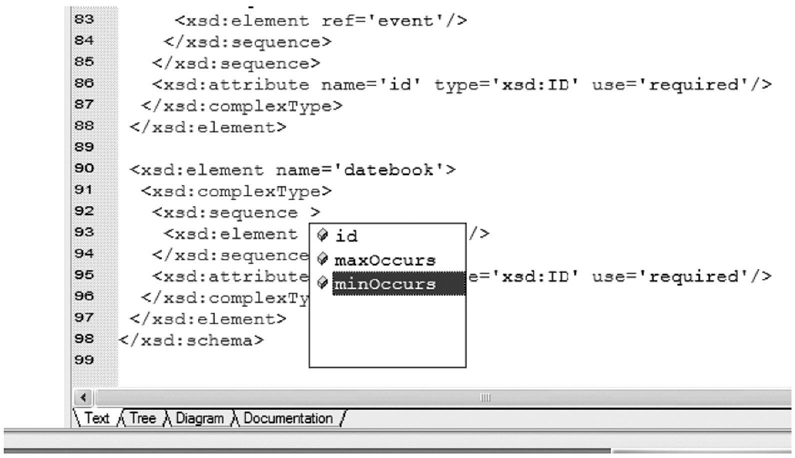
```
<xsd:element name='datebook'>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref='schedule'/>
   </xsd:sequence>
   <xsd:attribute name='id' type='xsd:ID' use='required'/>
  </xsd:complexType>
 </xsd:element>
```

The body of the element definition here indicates that a datebook object is a complex type (it is made up of other elements or attributes rather than just scalar data). In this particular case, the `<datebook>` includes both a sequence of `<schedule>` elements and an attribute, `id`, of type `xsd:ID`. The latter attribute is something of a guess on the part of the schema generator — in general, attributes that are named id are assumed to be identifiers, and consequently are always typed as both `xsd:ID` and made mandatory in their usage. In other words, the parser will treat it as a unique string (relative to the document) and if the id is not there then any validation will fair.

It's worth understanding what a sequence is to appreciate what's actually going on with the `<schedule>` element. The default operation that the Studio schema generator assumes is that every child element of a given element appears once and only once, and furthermore the order that the items appear is significant. Thus, with the way the schema currently works, only one `<schedule>` can appear within a `<datebook>`.

However, in practice, a date book may contain any number of schedules. For instance, one particular date book may contain three distinct schedules, one for each track of a conference. As another example, a date book may be included as part of an RSS news feed, and so may have dozens of schedules for completely different events, or no schedule whatsoever (perhaps nothing's scheduled for that week). In order for the schema to reflect this, it's necessary to modify it.

Fortunately, Stylus Studio has a number of ways to do this. If you know XSD reasonably well, the easiest way to make this change might be to modify the text file itself. Stylus Studio uses a form of keyword pop-up help called Sense:X. In essence, the editor keeps track of its current context, and provides the elements or attributes in a pop-up list that are defined by an internal schema for the type of document you're editing. For XSD, you can add a space after the `<xsd:sequence` element above and Studio will show you the attributes available, as shown in Figure 5.



```
      <xsd:element ref='event'/>
     </xsd:sequence>
    </xsd:sequence>
    <xsd:attribute name='id' type='xsd:ID' use='required'/>
   </xsd:complexType>
  </xsd:element>

  <xsd:element name='datebook'>
   <xsd:complexType>
    <xsd:sequence >
     <xsd:element                  />
    </xsd:sequence
    <xsd:attribute                 e='xsd:ID' use='required'/>
   </xsd:complexTy
  </xsd:element>
 </xsd:schema>
```

Figure 5. Sense:X feature shows available attributes

Selecting the minOccurs item will add this attribute, which determines the minimum number of times that the particular sequence can occur (in this case 0).

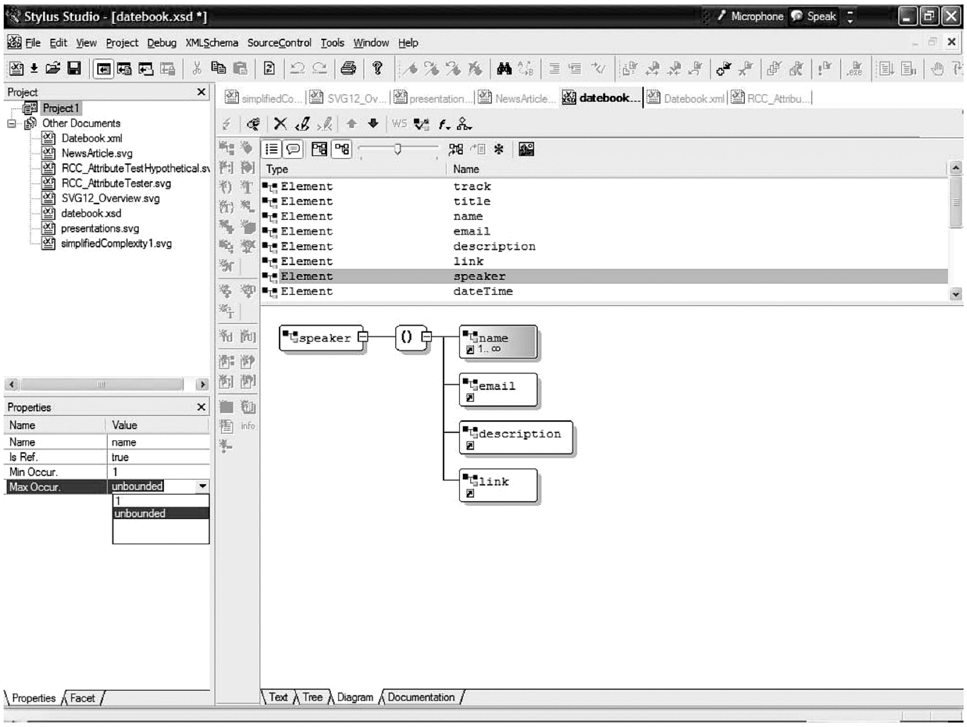
```
<xsd:sequence minOccurs="0">
```

Similarly, you can indicate that there are essentially an unbounded number of times that the sequence can occur, by using `maxOccurs="unbounded"`.

```
<xsd:sequence minOccurs="0" maxOccurs="unbounded">
```

There's an old joke that programmers know only three numbers — zero, one, and many. This has a certain basis of truth, especially with regard to schemas — if an item is included at all, it may be unique (occurs only once), but if it occurs twice then it probably can reasonably also occur three, four, one hundred, or ten million times. Hence, while its possible for a schema not to have one of the three values 0|1|unbounded, in practice it is unlikely.

An alternative way of changing this range is to use the graphic display of the schema. Switching to datebook.xsd, you can select the Diagram tab at the bottom of the XSD document editor to view a graphical representation of the schema. Clicking on the boxed plus signs for the datebook element will display the grouping indicator (given as a set of parentheses). Clicking on this will in turn show the properties associated with this group in the properties pane. You can then set the `minOccurs` and `maxOccurs` in the properties pane, taking advantage of the drop-down menus, as shown in Figure 6.


Stylus Studio - [datebook.xsd *]
File Edit View Project Debug XMLSchema SourceControl Tools Window Help
Project
Project1
Other Documents
Datebook.xml
NewsArticle.svg
RCC_AttributeTestHypothetical.sv
RCC_AttributeTester.svg
SVG12_Overview.svg
datebook.xsd
presentations.svg
simplifiedComplexity1.svg
simplifiedCo... SVG12_Ov... presentation... NewsArticle... datebook... Datebook.xml RCC_Attribu...
speaker
()
name
1..∞
email
description
link
Properties Facet
Text Tree Diagram Documentation

Type	Name
Element	track
Element	title
Element	name
Element	email
Element	description
Element	link
Element	speaker
Element	dateTime

Properties

Name	Value
Name	name
Is Ref.	true
Min Occur.	1
Max Occur.	unbounded

Figure 6. Setting the grouping properties manually

An alternative way of changing this range is to use the graphic display of the schema. Switching to datebook.xsd, you can select the **Diagram** tab at the bottom of the editor to view a graphical representation of the schema. Clicking the boxed plus signs for the datebook element displays the grouping indicator (given as a set of parentheses). Clicking the grouping indicator in turn shows the properties associated with this group in the properties pane. You can then set the `minOccurs` and `maxOccurs` in the properties pane, taking advantage of the drop-down menus, as shown in Figure 6.

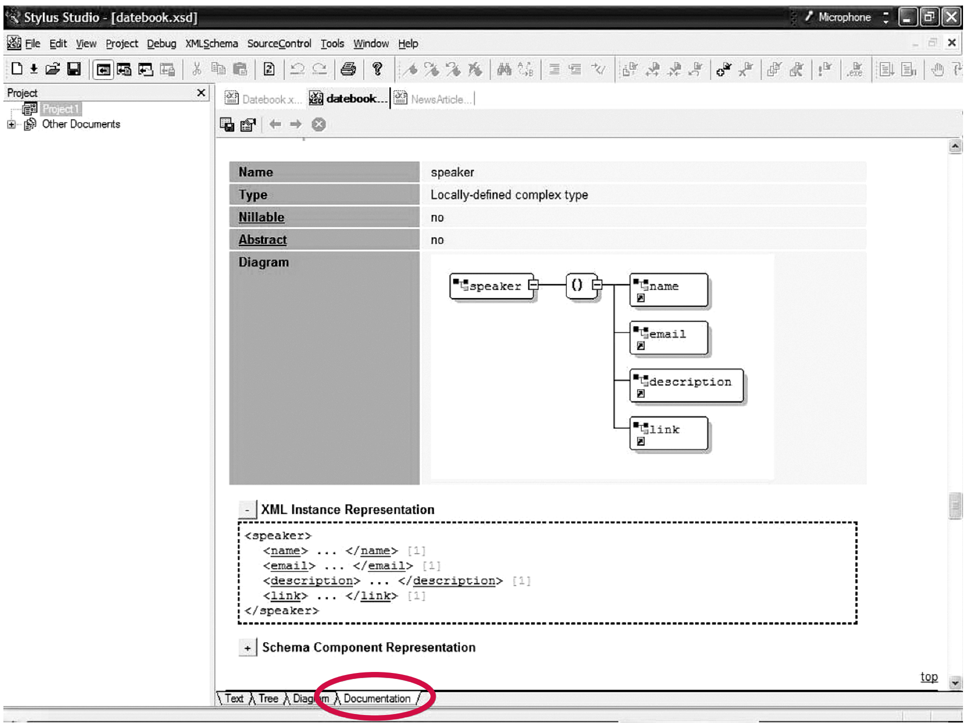
A similar adjustment needs to be made with the event items, since it is possible to have a schedule with no events (in essence, this would contain just a track label and a description of the schedule, perhaps in anticipation of additional data). Because the original instance contained only one schedule with multiple events, the generator assumed that a minimum of one and an unbounded maximum of events were possible, but you would need to deliberately set the minimum to zero in the schema:

```
<xsd:element name='schedule'>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref='track'/>
    <xsd:element ref='description'/>
    <xsd:sequence minOccurs='0' maxOccurs='unbounded'>
     <xsd:element ref='event'/>
    </xsd:sequence>
   </xsd:sequence>
   <xsd:attribute name='id' type='xsd:ID' use='required'/>
  </xsd:complexType>
 </xsd:element>
```

The rest of the schema could likewise be adjusted to take into account conditions that weren't obvious from the preliminary instance. Note that while it is generally recommended that you set such constraints as `minOccurs` and `maxOccurs` on `<xsd:sequence>` elements, even if these sequences contain only one item, you can set these attribute directly on the `<xsd:element>` elements as well.

This particular schema is fairly simple, in that most of the data types are simply strings. There are, however, two important exceptions: the `<dateTime>` and `<duration>` elements within a `<scheduledTime>` element. Because these fields need to store specific date or interval information, the schema must single them out for special treatment. As it turns out (more by linguistic convenience than anything else), there are two core data types defined within XSD called `xsd:dateTime` and `xsd:duration` that correspond to the conventions used earlier within this article. In addition to insuring that this data is valid, more advanced processing applications rely upon this data type to convert the string representation into internal date-time and duration data types used by the operating system. You can change these types in the **Text** tab using Sense:X, or in the **Diagram** or **Tree** tab using the property pane, as above.

Schemas are not necessarily easy to understand, especially when they begin to get into the dozens or hundreds of elements. One of the more useful facilities within Stylus Studio is the ability to view auto-generated documentation about the schema that you've created. The Documentation tab at the bottom of the editor can be used to view relevant information about specific elements in the schema, including their dependent child relationships, date type specifications, a representation of how an instance of that element would be used, and so forth. The documentation for the `<event>` element is shown in Figure 7.


Stylus Studio - [datebook.xsd]
File Edit View Project Debug XMLSchema SourceControl Tools Window Help
Project
Project1
Other Documents
Datebook.x...
datebook...
NewsArticle...
speaker
name
email
description
link
top
Text Tree Diagram Documentation

Name	speaker
Type	Locally-defined complex type
Nillable	no
Abstract	no
Diagram	

XML Instance Representation

```
<speaker>
   <name> ... </name> [1]
   <email> ... </email> [1]
   <description> ... </description> [1]
   <link> ... </link> [1]
</speaker>
```

Schema Component Representation

Figure 7. Documentation view of an element within Stylus Studio

Incidentally, directly above the documentation pane are two buttons, Save Documentation and Options, that can let you save the documentation for the schema as a Javascript enabled HTML page and determine which features will specifically appear within the documentation.

You can additionally add your own documentation to this mix via the `<xsd:annotation>` element. An annotation serves one of two purposes — either providing contextual help and descriptive information about the element or attribute in question (using the child `<xsd:documentation>` element), or providing application-specific information, such as binary code to describe how the element should be implemented in a given viewer. While the latter is beyond the scope of this article, the use of documentation can significantly increase the legibility of the schema.

To add an annotation in the **Text** tab to the `<event>` element, you would incorporate the following elements:

```
<xsd:element name="event">
  <xsd:annotation>
   <xsd:documentation>An event is one or more related blocks of time
with associated contact information.</xsd:documentation>
  </xsd:annotation>
  ....
  <xsd:complexType>
</xsd:element>
```

The `<xsd:documentation>` element contains the text you wish to add, with white space being preserved. There's nothing stopping you from making this information HTML or any other descriptive format, but Stylus Studio recognizes text only. However, once added, the descriptive comments contained within the `<xsd:document>` element appear as tool-tips whenever the mouse is over that element's box in Diagram view.

Obviously, even a tutorial article such as this cannot go into all the details of creating complex schemas, but with Stylus Studio it is fairly easy to experiment with schema elements to get a better feel for what does and does not work.

> VALIDATING AND USING SCHEMAS IN STYLUS STUDIO

While generating schemas within Stylus Studio is both easy and comprehensive, you can also use the schemas in question to significantly improve your own development experience with Stylus Studio, both in terms of validating schemas and in developing your own schema instances.

Validation is the process of ensuring that an XML instance that you create conforms to the Schema. On a development basis, validation can be used to check that the data is within the bounds of being legal, a very important consideration because invalid data can potentially break applications and cause all kinds of havoc. Note that a valid schema instance is not necessarily a legitimate document. For example, if you fail to specify in your invoice schema that the number of line items to be ordered is less than 100, then a malicious coder could dummy up an instance asking for 1,000,000,000 items and the computer would merrily attempt to fill all one billion orders. However, validity can help catch a lot of mundane errors if used properly.

If you have an existing schema and you want to bind an XML document to that schema in Stylus Studio, choose **XML->Associate XML with Schema** from the application menu. This displays a standard File Open dialog, letting you selected the schema or DTD that you want to associate (also known as bind) to the document. Note that the default option in the dialog is to bind a DTD, so you will need to explicitly set this to XSD schema in order to display any such schemas in the file system.

Once a schema is bound to a document, you can validate the document by selecting **XML->Validate Document** from the application menu. If the document is valid, a pop-up appears indicating this fact. If it's not, then all of the points indicating where the document is not valid are displayed in the Output Window, including the line and character position where the parser found the problem.

Perhaps the most common validation errors occur when a sequence of items in the instance are out of order with respect to the schema — for instance, rather than the `<scheduledTime>` element having the items in the order:

```
<scheduledTime>
    <dateTime>2003-01-20T12:00:00</dateTime>
    <duration>PT1H</duration>
</scheduledTime>
```

the order of the two children are reversed:

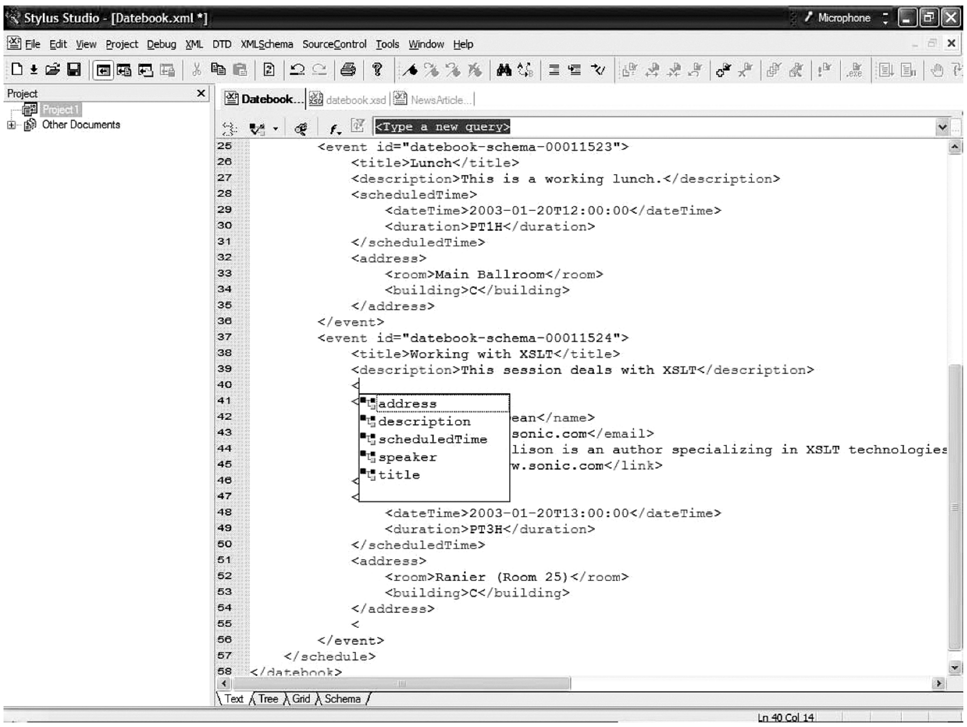
```
<scheduledTime>
    <duration>PT1H</duration>
    <dateTime>2003-01-20T12:00:00</dateTime>
</scheduledTime>
```

This most typically produces a response such as:

```
file://c:\Publishing\Sonic\Datebook.xml:18,18: Element 'duration' is
not valid for content model: '(dateTime,duration)'
The XML document Datebook.xml is NOT valid (1 errors)
```

These types of errors can be difficult to resolve, especially when dealing with fairly complex schemas. Pay close attention to the document, line and character positions displayed in the Output Window, and check to make sure that the order of the elements or attributes are correct.

One real benefit to binding a schema to a document is that Stylus Studio can use that schema to drive the Sense:X features for that document — in other words, Stylus Studio displays hints about what elements are available within the context of a specific element, what attributes are associated with each element, and so forth. An example of this can be seen in Figure 8, which shows insertion of event children.



```
            <event id="datebook-schema-00011523">
                <title>Lunch</title>
                <description>This is a working lunch.</description>
                <scheduledTime>
                    <dateTime>2003-01-20T12:00:00</dateTime>
                    <duration>PT1H</duration>
                </scheduledTime>
                <address>
                    <room>Main Ballroom</room>
                    <building>C</building>
                </address>
            </event>
            <event id="datebook-schema-00011524">
                <title>Working with XSLT</title>
                <description>This session deals with XSLT</description>
                <
                <
                                    ean</name>
                                    sonic.com</email>
                                    lison is an author specializing in XSLT technologies
                                    w.sonic.com</link>
                <
                <
                    <dateTime>2003-01-20T13:00:00</dateTime>
                    <duration>PT3H</duration>
                </scheduledTime>
                <address>
                    <room>Ranier (Room 25)</room>
                    <building>C</building>
                </address>
                <
            </event>
        </schedule>
</datebook>
```

Figure 8. Binding a schema to an instance makes it possible to use Sense:X

> SUMMARY

XML based schemas are very quickly emerging as one of the most important facets of the XML revolution. Schemas make it possible to both provide the full “legal” structure of an XML document and to specify type characteristics for data contained within the document. A schema is the set of rules that defines how a document is put together; using Stylus Studio, it’s easy to put together these rules with relatively little hassle, and to take a schema from an idea to a full-blown implementation in a very short duration ... PT1H or less!

ABOUT SONIC SOFTWARE CORPORATION

Sonic Software provides the first comprehensive business integration suite built on an enterprise service bus (ESB). The Sonic product line delivers a distributed, standards-based, cost-effective, easily managed infrastructure that reliably integrates applications and orchestrates business processes across the extended enterprise. Sonic is the world's fastest growing integration and middleware company and counts global leaders among over 500 customers in financial services, energy, telecommunications and manufacturing. Sonic is an independent operating company of Progress Software Corporation NASDAQ: PRGS), a $300 million software industry leader. Headquartered in Bedford, Mass., Sonic Software can be reached on the Web at www.sonicsoftware.com, or by phone at +1-781-999-7000 or 1-866-GET-SONIC.

ABOUT SONICMQ

SonicMQ is the industry's most scalable enterprise message server, delivering exceptional reliability, extensive connectivity, unmatched management capabilities and comprehensive security for business-critical communication across the extended enterprise.

Corporate and North American Headquarters

Sonic Software Corporation,14 Oak Park, Bedford, MA 01730 USA

Tel: 781-999-7000 Toll-free: 866-GET-SONIC Fax: 781-999-7202

EMEA Headquarters

Sonic Software (UK) Limited, 210 Bath Road, Slough, Berkshire SL1 3XE, United Kingdom

Tel: + 44 (0)1753 217000 Fax: + 44 (0)1753 217001

sonic SOFTWARE™

www.sonicsoftware.com