

## **Fabula**

### **bi-lingual, multi-media, children's storybook software based on XML and Mozilla**

**Eoin Campbell**

XML Workshop Ltd., Dublin, Ireland

ecampbell@xmlw.ie

<http://www.xmlw.ie>

**Brian King**

XML Workshop Ltd., Dublin, Ireland

bking@xmlw.ie

<http://www.xmlw.ie>

#### **Abstract:**

*Mozilla (Netscape 5.0) provides an excellent platform on which to develop cross-platform XML applications. Mozilla provides a very complete and sophisticated foundation on which to start building a Fabula Reader and Maker. Much of the required functionality is built-in, allowing the main effort to be focused on adding new Fabula-specific features.*

### **The Fabula project**

The Fabula project was conceived, and is managed, by the Reading and Language Information Centre at the University of Reading, in the UK. It is funded under the EU 4th Framework research programme, sponsored by the European Commission, and commenced in July 1998.

A need was identified to provide computer-based software to aid in the learning of lesser-used languages in schools throughout Europe. These languages suffer due to their minority status, both in the numbers of people that speak them and through public exposure in Europe, in the area of computer aided learning. The goal is to provide free software that allows teachers, linguists, and children to create and distribute multi-media stories. The status of the languages will be raised among children if learning is available through computers. Research in this area found that language learning was enhanced when it varied from the traditional classroom methods, in this instance through computers. Computers open up a new world of interaction for children, and make learning fun.

The target children's activity that the software is intended to address is the creation and reading of stories. There are two components in Fabula, an authoring tool (called the Maker), and a story interaction tool (called the Reader). The Reader is more than just a viewing tool, as the story can be narrated on audio, word-games can be played, and there is a built-in mini-dictionary.

The people involved in Fabula are made up of the 'core partners' and the 'evaluation partners'. The core team's role is in the design, development, and piloting of the software. The evaluation partners organise and manage in the distribution and evaluation of the software in schools in participating countries, and are listed below.

#### **Core partners**

University of Reading, UK (Reading & Language Information Centre, Department of Typography & Graphic Communication)

University of Brighton, UK (School of Information Management, School of Languages)

DTP Workshop, Dublin, Ireland

#### **Evaluation partners**

Friesland: Gemeenschappelijk Centrum voor Onderwijsbegeleiding fryslân.

Ireland: Institiúid Teangeolaíochta Éireann.

Basque region: Kanboko Ikastola, France.

Wales: St Helen's Welsh Centre, Swansea.

Catalonia: University Pompeu Fabra, Barcelona.

## History

The first Fabula project meeting was held in Reading (UK) in October 1998, where two workshops were held. The translation workshop involved the evaluation partners and their translators, and involved each partner translating the chosen prototype story (“A Lovely Bunch of Coconuts”) into their own lesser-used languages. Once the initial translation was completed, the language used in each story was simplified so that it was suitable for children learning the language as a second language, rather than their first language.

The technical workshop discussed the overall architecture and goals of the software, and identified the main features required.

Between October and March of 1999, the main work concentrated on building a prototype of the Fabula Reader (for viewing stories) using Macromedia Director, developing the Fabula website (<http://www.fabula-eu.org>), and writing the various reports required by the project sponsors.

A second Fabula project meeting was held in Dublin in March 1999, and again consisted mainly of two workshops. The evaluation partners reviewed a prototype of the Fabula Reader, and the core partners finalised the detailed feature set for both the Reader and Maker, and made a final decision on the implementation strategy.

From April until December 1999, development of the software based on Mozilla was carried out in Brighton (Reader) and Dublin (Maker).

The third project meeting was held in Barcelona in November 1999, where the evaluation partners first got a chance to use and review the real working Fabula software, and learn how to create and view their own stories.

In January 2000, prototype software, in each of the 9 project languages (Reader) and the 4 major languages (Maker), was released to the evaluation partners for testing in schools.

The final project meeting will be held in May 2000 in Bayonne, where experiences and lessons learned from the schools evaluation phase currently underway, will be shared back to the developers, and a set of changes to the software agreed, prior to formal release at the end of the project in June 2000.

## Implementation

During the initial design stage of the project in the latter part of 1998, the various potential implementation strategies for the software were examined. At that time, the strategies available included the following options:

- Complete development using Java

- Complete development using C/C++

- Development of Fabula software on top of existing browser platform (Mozilla or Internet Explorer)

Two major factors influenced the final choice of development strategy. First, schools are generally poorly equipped with computers, so the software had to run reasonably well on older, slower machines, such as were available in 1995. Second, the amount of time available for development was approximately 18 man-months in total, so in order to release actual working software within the lifetime of the project, every opportunity to build on existing efforts rather than develop from scratch had to be taken. A third factor was the desire for a cross-platform solution, supporting both the Windows and Macintosh platforms.

## **Java**

The Java approach was rejected in spite of the current fashion for the language and technology. In principle, Java offers the advantage of cross-platform binary compatibility. The price of this compatibility however, is that only modern computers will run Java programs at acceptable speeds. Such computers are simply not available in the typical European primary school. In addition, anecdotal evidence is that there are subtle variations (and unsubtle bugs) both in Java compilers used on different platforms, and in Java Virtual Machines on different platforms.

## **C/C++**

Development in C++ would avoid the performance problems of Java, and many off-the-shelf libraries are available to implement specific required components of the software. However, one of the aims of the project is to support both the Windows and Macintosh (and possibly Linux) operating systems, and most component libraries (e.g. Microsoft Windows Foundation Classes) are limited to a single platform. This would have meant multiple parallel efforts to develop the User Interface in particular, on each platform. For these reasons, C++ was also rejected.

## **Browser software**

Both of the major browsers that predominate on the desktop were candidates for selection. Beta previews of Internet Explorer 5.0 were available in late 1998, which showed the extent of support for XML and its related technologies within that browser, and the user interface is also quite customisable. However, Internet Explorer was ultimately rejected both because of concern about when version 5 would become available on the Macintosh (justifiably, as it was only released in March 2000), and the fact that little editing capability is built into the browser.

Mozilla software and source code has been publicly available since April 1998, when the source code for Netscape Communicator 4.04 is the name was released into the public domain. Mozilla seemed ideal for the Fabula project, as it was free, cross-platform, based on C++ (and therefore reasonably fast), supported XML technologies, and offered the basis for both the editing and viewing functionality required.

On the other hand, its future has never been very certain, and the rate of progress towards a formal release of a version 5 browser has been sedate, to say the least. In its early days, plans to continue development of the Composer HTML editing component were dropped in favour of concentrating entirely on the browser, meaning that considerable work would be required to provide this functionality for Fabula.

However, in early 1999, Mozilla started the 'Ender' project, which was a replacement for Composer, and this, together with the strong commitment to support for W3C standards in general, and the innate technological attractiveness of the whole Mozilla endeavour, led the designers to commit to using the Mozilla source code (all 65 Megabytes of it) by March 1999, when the development of the software began in earnest.

## **Mozilla overview**

The Mozilla software architecture is reasonably well documented, and a book on the topic has already been published. The architecture is highly modular, and consists of many separate components, such as the layout engine, the network library, the JavaScript engine, etc. The core presentation software is the 'Gecko' Layout Engine, which implements the HTML4, CSS1, and XML 1.0 Recommendations of the W3C. A feature of Mozilla is that the layout of the entire User Interface, not just Web pages, is implemented using Gecko.

The User Interface component of Mozilla is called the XPFE (Cross-Platform Front-End) . Originally, Netscape maintained multiple parallel development streams to implement Navigator on the major different platforms of Windows, Macintosh and Unix. After the software was released, an ambitious project to develop a cross-platform UI architecture began, in order to avoid this duplication of effort. This resulted in the XUL (Extensible User-Interface Language) pronounced 'zool', which is a mechanism for defining standard UI components such as menus, dialog boxes, trees, lists, etc. using a mixture of well-formed XML, based on RDF ideas. A particular set of user interface objects is called a 'chrome' in Mozilla terminology, and it is intended to soon be possible to maintain many different chomes for the browser, and switch dynamically between them.

Overall, the XPFE project has been extremely successful, and it is possible to develop complete cross-platform applications without the need for any OS- or Window System-specific knowledge or programming. This seems to fulfil much of the original promise of Java, without the performance hit.

## **Fabula software**

The Fabula project consists of an authoring application, a viewing application, and stories written in the Fabula Markup Language.

The Fabula Reader is based directly on the standard Mozilla browser binary. Instead of the typical browser User Interface, however, a Fabula UI has been developed using XUL as a definition language. All interactions are programmed using JavaScript only. This means that the Reader can be maintained as an alternative 'chrome' (User Interface) on top of the standard browser distribution. The Fabula Reader is quite easily extensible by non-traditional programmers, without any need for special development software environment, and we hope that interested parties will add more features to the current interface. It was not specifically designed to be this way, but the Mozilla architecture allows it.

The Fabula Maker is based on the Mozilla 'Ender' HTML editors project, but has required some changes to the C++ code. Essentially, the Mozilla editor has a built-in knowledge of all the defined HTML elements. The changes at this level involved removing many of those elements, and adding new elements specific to Fabula. In addition to these changes, most of the work on the Maker, like the Reader, has involved developing a specific UI using XUL and JavaScript to implement the features required to allow children and teachers to easily create and save stories in a WYSIWYG environment.

Fabula stories are a collection of text, images and sounds. Stories are intended to be written in language pairs, such as English-Irish, Spanish-Catalan, or Dutch-Frisian, and the complete story text is stored for both languages in the same file. Because the Fabula Reader currently uses CSS rather than XSLT for presenting stories, the XML used to represent stories is well-formed rather than valid, to overcome limitations of CSS in presenting images (among other things). When Mozilla provides full support for XSLT (some prototype work has already been done), then it will be possible to apply a formal XML DTD to stories. This will also simplify the XML markup generated by the Maker.

## **Story content**

This section explains the details of the Fabula Markup Language as it is currently used in stories.

```
<page hidden="true" class="pl2">
<lang1>
<audio xml:link="simple"
      href="javascript:playSound('file://sounds/fy/fy_05.wav')"/>
```

```

<game xml:link="simple" href="javascript:playGame('p05.101')"/>
<sentence>'Dat is alles wat er hat,' sei de keninginne.</sentence>
<sentence>'Dat is alles wat ik ha wol,' sei de kening.</sentence>
</lang1>

<lang2>
<audio xml:link="simple"
      href="javascript:playSound('file://sounds/nl/nl_05.wav')"/>
<game xml:link="simple" href="javascript:playGame('p05.102')"/>
<sentence>'Dat is alles wat hij heeft,' zei de koningin.</sentence>
<sentence>'Dat is alles wat ik wil hebben,' zei de koning.</sentence>
</lang2>

<picture><html:img src="images/05.jpg" /></picture>

</page>

```

**Figure 1. XML code sample**  
*Sample of XML story code*

This code example illustrates that the elements give the story a particular meaning. The `<lang1>` and `<lang2>` elements represent the two language panels. Within each one is contained an image. Each language panel contains an optional sound element, 'audio'.

## Layout

Style sheets are mechanisms for organising the layout and appearance of XML documents. The XSL (XML Style Language) is currently not widely implemented and is not available yet in the Mozilla project. It was therefore deemed unsuitable for the Fabula story layout.

The CSS (Cascading Style Sheets specification) was chosen for the Fabula project. Mozilla has committed to full support for CSS Level 1 and some support for CSS 2. The benefits of using CSS are:

- The separation of style and content. The styles for the Fabula story elements are located in an external, linked style sheet.

- The syntax is easy to understand

- It can be manipulated using JavaScript.

```

page {
  display: block;
  position: absolute;
  background-color: white;
  border-style: solid;
  border-color: black;
  border-width: thin;
}
page[class=pl1] lang1 {
  display: inline;
}

page[class=pl2] lang1 {
  left: 50.25%;
}
sentence {
  margin-left: 3%;
}

```

```

margin-right: 3%;
display: block;
color: black;
cursor: default;
}

```

**Figure 2. CSS code sample**

*Example Style Definitions from the Fabula Story Stylesheet*

**Some of the main properties include:**

display - the type of box used to render the element.

position - of the element. An absolute positioned element does not appear within the normal flow of the document (as relative positioned elements do), but in a fixed area defined by the sizing.

width / height - percentage values ensures that the elements appear in the same size ratio on re-sizing of the window. The root <story> element has a size of 'screen.height' and 'screen.width'. This ensures that the proportional display of the story is consistent across platforms using different screen resolutions.

border-<property> - manipulation of the style, colour, and size of a border around a particular element.

left - distance of element from left-most point of containing element

top - distance from top of containing element.



**Figure 3. Fabula styles**

*Layout of a page in the Fabula Reader*

The styles are attached to the story using the External Style Sheet Declaration, as illustrated with the following syntax:

```
<?xml-stylesheet type="text/css" href ="story.css"?>
```

## Linking

Linking is one of the key concepts in a Fabula story. It is used for playing sounds, activating the wordgame and for connecting words. The XLink (XML Linking Language) , provides a more flexible linking mechanism than standard HTML hyperlinks.

HTML contains a link element called the Anchor tag (<A>). This can have two forms, either a marker in text to indicate a location that can be pointed to, or a pointer which points to a location, usually a filename, but also a location within a file.

```
HTML Marker: <A NAME="section3">
HTML Pointer: <A HREF="#section3">
```

HTML offers only a simple, one-way link. A can point to B. B has no inherent knowledge of where A is, or even if there are a number of other pointers C, D, and E, which also point to B. In the Fabula context, a more powerful linking mechanism is required, which supports two-way and multi-point links. XLink is a draft proposal of the W3C that addresses the needs complex definition of relationships between objects.

XLink is the appropriate linking mechanism in Fabula because:

- It provides a defined set of Fabula linking tags, due to the fact that it is native to XML, the story implementation mark-up.

- Multi-directional links can be accommodated

- The application can choose its own behaviour for the link.

```
<page hidden="true" class="pl2">
<lang1>
<audio xml:link="simple"
  href="playSound('file://sounds/s02fy.wav')"></audio>
<game xml:link="simple" href="javascript:playGame('p2.11')"></game>
<sentence>Oan de oare kant fan it wetter libbe in
  <marker id="l.p02.m01" xml:link="simple"
    href="javascript:lightMarker('l.p02.m01')">
    ynklauwerige</marker> kening.</sentence>
<sentence>Hy seach alle dagen nei it eilantsje.</sentence>
</lang1>
<lang2>
<audio xml:link="simple"
  href="playSound('file://sounds/s02nl.wav')"></audio>
<game xml:link="simple" href="javascript:playGame('p2.12')"></game>
<sentence>Aan de andere kant van het water leefde een
  <marker id="l.p02.m02" xml:link="simple"
    href="javascript:lightMarker('l.p02.m02')">
    hebberige</marker> koning.</sentence>
<sentence>Hij keek elke dag naar het eilandje.</sentence>
</lang2>
<picture> <html:img src="images/02.jpg"/></picture>
<linkgroup>
  <linkset>
    <pointer idref="l.p02.m01" />
    <pointer idref="l.p02.m02" />
  </linkset>
</linkgroup>
</page>
```

**Figure 4. XLink mark-up**  
*Example story mark-up containing XLink links*

The above mark-up can be described as follows:

The <game> element, when used in the story, sets the word game in an active state. This is represented in the story layout as a graphic that appears in both of the language panels. It highlights the participating words in the corresponding panel. The <marker> element contains the word or words that are participating in the wordgame. Each marker has a unique identifier, to make it easy to specify pointers to that piece of text.

The actual links, or definitions of relationships between pieces of text, are not defined within the text, but stored separately, in the <linkgroup> section of the page. The separation of the links from the locations is considered good practice in XLink, and offers a number of advantages, including simplicity of the text. Within the <linkgroup> element, the definition of each of the individual links on the page takes place. For Fabula, we consider only the case of in-page links: intra-page links will not be supported.

The <linkset> element contains the definition of a single link.

In this example, each link is a simple bi-directional link between two and only two marked locations. However, more complex links can be defined. Whether these will be supported in Fabula will depend on the level of implementation reached in both the Maker and Reader. Each <pointer> element defines one endpoint of a link.

The software is only using simple XLinks, and not currently harnessing the more powerful extended links.

## **User interface**

We chose to adopt the method used in-house in Mozilla to define and create our user-interface (UI). This is the Extensible User-Interface Language (XUL). This technology is an XML application, and is still evolving. It will enable ease of creation and customisability for the Fabula interface. The code does not need to be compiled with the rest of the code base. It harnesses the power of the existing 'Gecko' layout engine, which is used for displaying XML and HTML content.

XUL (pronounced 'zuul') is a mark-up language to define the various widgets and elements that make up a user interface. In effect, interface descriptions are stored as individual files. The descriptions may be in a single file or spread over multiple files.

"Somewhere accessible, stored in an application specific form, are hunks of UI description—streams of XML—corresponding to hunks of UI machinery. The app might store these descriptions as individual files; as resources; as database entries; or even remotely to be accessed through URLs."

Grouped together, the user interface elements, resources and files are collectively known as the "chrome".

## **Benefits**

The primary feature of XUL is the separation of interface from functionality. Though not a new concept, it is very successful in achieving this.

It is cross-platform. There will be some platform specific information and resources to obtain the necessary appearance but the Macintosh and the Windows versions of the Maker chrome (and the Reader, the interface of which is also being developed with XUL) will share many common interface files. This saves on overhead development when changes need to be made, and could be advantageous if Fabula is developed for other platforms at a later date.

Here are the most common features of the XUL interface, with examples. It is worth noting that XUL files can have style sheets applied to them, to define such things as size, colour and behaviour of the various items.

## Menu items

The interface elements are descriptive, similar to the story elements contained in the storybook XML document. A “menu” contains a set of “menuitem” elements, which are wrapped in a “menupopup” element. This will appear as a standard drop down menu within the application.

```
<menu value="&makerinsertMenu.label;"
      accesskey="&makerinsertmenu.accesskey;">
  <menupopup>
    <menuitem id="makerNewPage" accesskey="1" key="makernewpgkb"
              observes="Maker:NewPg" />
    <menuitem id="makerSound"      accesskey="2" key="makersoundkb"
              observes="Maker:Sound" />
    <menuitem id="makerImage"     accesskey="3" key="makerimagekb"
              observes="Maker:Image" />
  </menupopup>
</menu>
```

**Figure 5. XUL mark-up**

*Example mark-up for Fabula Maker 'Insert' menu*

The behaviour of the menu item is defined by a call to a JavaScript function contained in an externally linked file. This can in turn behave a number of ways, such as activating a portion of the C++ code that carries out a particular task, or manipulating the story using the Document Object Model. Each “menuitem” behaves a certain way, and is determined by the "onaction" event. This is set in the “broadcaster” element, which is linked to the “observes” attribute.

```
<broadcaster id="Maker:Sound" value="&makerinsertSoundCmd.label;"
             oncommand="insertSound()" />
```



**Figure 6. Menu**

*'Insert' menu from the Fabula Maker*

## Toolbars

The flexibility of XUL enables a break from traditional designs of user interfaces. The Fabula Reader contains two vertical toolbars, while the Maker has one vertical toolbar, located on the left side of the content area. It consists of a number of buttons that carry out some of the most common tasks of the software.

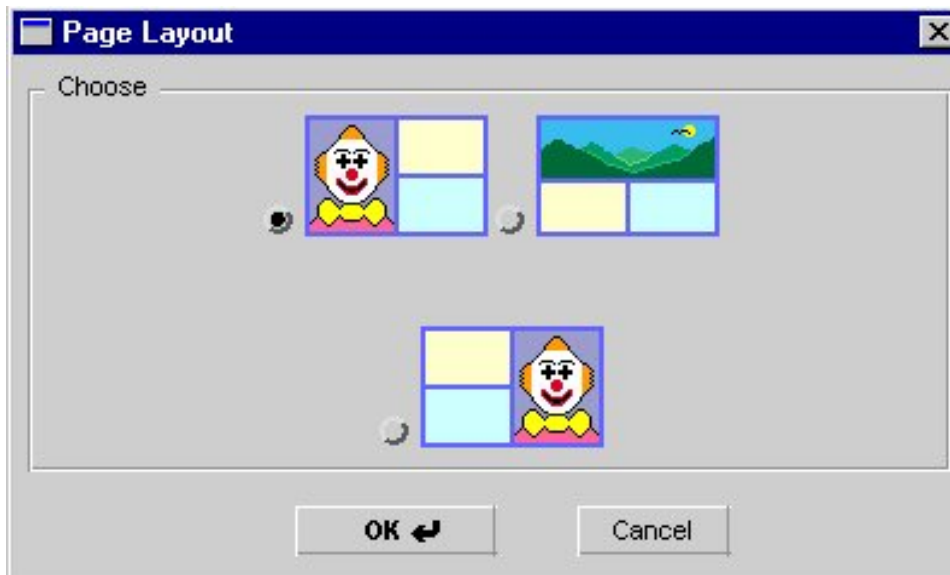
```
<toolbar id="MakerToolbar" class="maker" align="vertical" >
<button button class="toolbar left" orient="vertical"
  id="makNewButton" observes="Maker:New" />
<button button class="toolbar left" orient="vertical"
  id="makOpenButton" observes="Maker:Open" />
<!-- Comment: more menu items go here -->
</toolbar>
```

**Figure 7. Toolbar mark-up**  
*Toolbar from the Fabula Maker*

## Dialog boxes

Dialog boxes will be an integral part of the Fabula interface, as with many applications. The purpose of a dialogue will be to present the author of the story with options when carrying out a particular task.

Each dialog is laid out as a separate XUL file, and styled via CSS.



**Figure 8. Sample dialog box**  
*'Insert - New Page' dialog box from the Fabula Maker*

## JavaScript — module interaction

JavaScript is a Netscape developed scripting language and a scaled back object-oriented language. It is similar in syntax and structure to C/C++ and Java, though apart from these similarities, they are unrelated languages. It is a superset of the ECMA-262 Revision 3 (ECMAScript).

If we consider each element of the Fabula software; the story, the user-interface, the C++ code base as a distinct module, then the main tool of interaction between the modules is JavaScript.

It is used by the XML story within the Reader to carry out such functions as playing a sound or activating a wordgame.

It is used by the XUL user-interface to carry out simple document operations like changing the style properties or adding a new element via the DOM API.

It is used as the primary interface into the C++ code to carry out complex operations such as saving the story to disk, or inserting an image.

```
function ShowCoverPage()  
{  
    var coverPage =  
        window.frames[0].document.getElementsByTagName("cover");  
    coverPage[0].setAttribute("hidden", "false");  
}
```

**Figure 9. JavaScript**  
*Example JavaScript function*

This function, when called, returns the user to the cover page of the story. It uses the DOM method – `getElementsByTagName` – to get a handle on the cover element in the story, and it turns off its hidden attribute, thus making it visible.

## Localisation / internationalisation

### **There are three types of files that need to be localized in Fabula:**

The DTD (\*.dtd) files.

The DTD files each contain a list of entities that need to be localized. This is an example:

```
<!ENTITY windowTitle.label          "Insertar sonidos">
```

The Property (\*.properties) files.

Property files are used for storing strings that are used in JavaScript and C++. Each string has a name which is associated a string through the '=' operator.

```
SelectSoundFile=Select Sound File
```

This is in turn accessed through an API call, specific to the module you are working in. This is how it would be done, in JavaScript, for the Maker:

```
var select = editorShell.GetString("SelectSoundFile");
```

The chrome registry

The chrome registry is a file that contains the paths relating to the location of the various chrome modules. This is currently being re-architected, and there will soon be in place a mechanism for switching between 'skins' (chrome packages) dynamically.

We are able to leverage tools that have been developed by third party groups to automate as much as possible this process. One of these tools is the 'MozillaTranslator'. This enables you to extract the strings from the DTD and Property files into a single glossary. This means that a single file can be sent off to your localisation agent for translation. On return of the localised strings, they can be imported into new locale files.

## Status

Alpha prototypes of the Fabula Reader and Maker are available. There is currently an evaluation taking place in schools in participating countries, and the next 'beta' release will take into account findings from this evaluation.

The software is due for full release in June. It will be available on CD, or via download from the Fabula website. A sample storybook - 'A Lovely Bunch of Coconuts' by Dennis Reader - will be published in all five of the minority languages, with a combined print run of approximately 15,000.

## Acknowledgments

A useful introduction to the Mozilla project and the source code is available in the following book:

Netscape Mozilla Source Code Guide by William R. Stanek

Netscape Press; ISBN: 0764545884

## Bibliography

[XPToolkit]      (Introduction to the XPToolkit Architecture -  
<http://www.mozilla.org/xpfe/xptoolkit/introduction.html> )

## **Authors**

### **Eoin Campbell**

Technical Director  
XML Workshop Ltd.  
Postal Address:  
10 Greenmount Industrial Estate, Harold's Cross  
12 Dublin  
Ireland  
Telephon: +353 1 4547811  
Fax: +353 1 4731626  
E-mail: [ecampbell@xmlw.ie](mailto:ecampbell@xmlw.ie)  
Web: [www.xmlw.ie](http://www.xmlw.ie)

**Eoin Campbell** – Eoin Campbell founded XML Workshop in 1998, following a number of years as an independent consultant. He has spoken at the following industry conferences: SGML '98.

### **Brian King**

Software Engineer  
XML Workshop Ltd.  
Postal Address:  
10 Greenmount Industrial Estate, Harold's Cross  
12 Dublin  
Ireland  
Telephon: +353 1 4547811  
Fax: +353 1 4731626  
E-mail: [bking@xmlw.ie](mailto:bking@xmlw.ie)  
Web: [www.xmlw.ie](http://www.xmlw.ie)

**Brian King** – Brian King joined XML Workshop Ltd. in early 1999, and has worked on the Fabula project for 1 year.