Final Specification

For

XMP Blaster 2005

Prepared By Quinn Daly

The Thomas White Photography Team
(Preston Bryant, Quinn Daly and Jonathan Janssen)

Colorado School of Mines

Summer Field Session

May 2005

Index

Abstract	1
Definitions	1
Introduction to the Product	1
Requirements and Specifications	2
Design or Solution Approach	3
Implementation Details and Results	5
Conclusions and Future Directions	5

Abstract

Before digital cameras came into play, photographers would take the pictures that they liked and scan them onto their computers as a digital format of some kind. As a result, a photographer may take a thousand pictures, but only scan fifty of them. As the digital camera becomes more popular, we see a drastic increase in pictures stored on the computer. Those thousand pictures that a photographer may take might now be easily stored on the computer. With this influx of digital images stored on machines, it is necessary to organize these pictures using metadata.

With the use of metadata and digital cameras, it is now more convenient for a photographer to use templates to fill in the information contained in a picture. In many cases, a photographer will want to prepare these templates before a shoot. For instance, a photographer knows that he will be shooting a basketball game between team A and team B on a certain date. He can fill in that information ahead of time, and apply it to all pictures when he downloads them to his machine.

Our job is to provide the client with tools that will help photographers to manage custom templates. Our goal is to make it easier for photographers to create "clean" templates for their shoots using any standard or custom schemas (empty templates).

Definitions

Metadata—Data about data. In this case, metadata is extra information attached to a picture file. For instance, the date that the picture was taken is part of the picture metadata. This data is imbedded into the picture.
XML—Extensible Markup Language
XMP—Extensible Metadata Platform
EXIF—Exchangeable Image File
IPTC—International Press Telecommunications Council

Introduction to the Project

A user can create a custom template without using this product. When a user wants to create a template in Adobe Bridge, he can do so by accessing the pictures File Info Panel, filling in the information, and saving the data as a template. So if a user can create a template using Adobe Bridge, why do we need to create an application to do the same thing? When a user creates a template using Adobe Bridge, the template file that is saved contains a lot of other information that the user does not enter in. For instance, any information that the camera automatically enters into the pictures file, such as picture

creation time and picture dimensions. Any information that is included in the picture file that you create the template from will be copied to any other pictures you apply the template to.

Because of this problem, Thomas White has asked us to create a program that creates "clean" templates, that is, templates that include only the information entered in by the user. Without these "clean" templates, the metadata in picture files would soon be unusable due to the erroneous data contained in the pictures.

The Template Maker application takes a list of all the standard schemas, along with any custom schemas included, and allows the user to choose which schemas to use. Once the schemas are chosen, the user then has the ability to fill in the data fields that belong to that schema. Once this is done, the user can save the template to the XMP Metadata Templates folder. From this point, whenever the user wants to use the template, he goes to the File Info Panel of the pictures to apply the metadata to and chooses the template from the list.

Requirements and Specifications

The Template Maker application is intended to work with the Adobe Bridge application as an add-in. The people at Adobe consider Bridge a development environment as well as an organizational tool. Bridge uses JavaScript to enable developers to create custom add-ins. This is what we had to use to create the product. Because we used JavaScript, our program was automatically cross platformed. There are still a few items that needed our help to make the program completely multi-platformed however, such as where to write files depending on what operating system is being used.

When developing the application, we had the option of either using the regular JavaScript add-ins for all of the dialogs, or using any web-based programming. This is possible because the Adobe Bridge browser window uses the Opera explorer. After much consideration, we decided that it would suite our needs better to use JavaScript for the entire application.

One big requirement of our application was that we needed to be able access the standard schemas as well as any custom schemas. This ultimately required us to consider a design approach that dynamically creates the Template Filler window.

The entire reason for creating this application was to create "clean" templates. Therefore, we needed to only include fields that were filled in by the user. Once the fields are filled in and the user is ready, the information needs to be stored as an XMP file, which is a specifically defined file that Adobe Bridge uses to embed the information. We needed to create the XMP file exactly as Bridge

creates its XMP files in order to get the information translated back into the File Info Panel windows.

One of the biggest requirements of this project was to get a solid product out. We were fortunate to be able to focus on one main task, which has helped us to come out with a product that we feel is worthy of use.

Design or Solution Approach

When considering the best way to approach the design problem for the Template Maker application, we had to keep in mind that we would not have a hard coded list of schemas. Because of this, we decided to read the schemas from a xml file that we created specifically for this application.

As one of the requirements of the application, the user should be able to choose what goes into the template file. Our design solution produces a dialog to allow the user to choose schemas (the schema chooser window), followed by a dialog that takes those schemas and creates a window using all of the schemas fields (the schema filler window). This window allows the user to fill in the schema information to create the template.

One important consideration to our design was the existing libraries that were made available to us by Bob Stucky, an Adobe programming guru. The main libraries that we use are the CheezyXMLParser and XmlNode. The CheezyXMLParser reads in an xml file and transfers the information into a tree node—the XmlNode object.

The XML file we are using to keep track of the schemas is carefully defined in order for us to easily use Bob Stucky's CheezyXMLParser. For each schema, we need a "schemapointer" tag. This tag will have attributes like the actual schema tag name, the namespace, if it is delete-able or not, and the schema description. The value for the tag will be the schema name as it will appear in the application, which also happens to be the tag name for the schema fields. Using the CheezyXMLParser, we can then find all of the nodes with the "schemapointer" tag. This is how we will populate the Schema Chooser window. From there, we can find all of the nodes with that value as the tag. For instance, the schema IPTC Contact will look something like this:

```
<schemapointer schema="Iptc4xmpCore" xmlns="http://iptc.org/..."  deleteable="false"
        description="IPTC Contact metadata">IPTC Contact</schemapointer>

<IPTCContact field="Phone" datatype="multiline">Phone Number(s)</IPTCContact>
<IPTCContact field="Email" datatype="multline">Email(s)</IPTCContact>
```

So we can grab all of the IPTCContact by taking out the space from the value of

the schemapointer tag and passing in that value to the root XmlNode object. The attributes for the IPTCContact and sibling fields are meant to facilitate the creation of the Schema Filler dynamic dialog and the XMP file. For instance, the field attribute must be the exact tag string as given in an Adobe Bridge template file. The datatype is used to create the different controls in the Schema Filler dialog.

The schema chooser is the first window that the user will see after selecting the add-in from the tools window in Adobe Bridge. It displays a list of schemas, allows the user to select the schemas he wants to use, and provides a general description of the schema when selected. We have decided to use two list boxes. One list box on the left to contain the list of all schemas in the xml file, the other to contain the schemas that the user has chosen. We are using two buttons in between the list boxes to help maneuver the schemas from one list box to another. When a schema is chosen, the user will select that schema in the left list box, click on the button with the caption ">>", and the schema will disappear from the left and appear in the right list box. We have include a description text that will be changed on the event of the left list boxes change event. This is located under the left list box, and the description is kept as an attribute in the xml file.

Because our development environment uses JavaScript, we were not able to create a website dynamically using the normal JavaScript controls that are common place in most websites. We were able, however, to use normal HTML controls. In order to create the website with the proper fields, the website would not generate until the user finishes choosing the schema's. At that point, the add-in would loop through each schema in the Schema Object, grab all of the fields and data types, and create static text for each field name, and the proper widget for each field type. For instance, if there was a field with a type Boolean, we could ignore the static text and create a checkbox with the text of the field name. The webpage design would most likely have all of the fields grouped by schemas and all on one continuous page. At the bottom of the page, we would have a Save button to save the template. The user would be able to choose where to store the template.

The JavaScript dialog approach does basically the same thing as the website approach. The dialog is generated dynamically once the user has selected the entire schema's to be used, with the field names generated as static text, and the field values generated as the proper widgets for their data types. The one difference from the HTML design approach is that the dialog contains a panel for each schema and a drop down box for the user to select what schema to fill in. All panels except for the one that belongs to the schema chosen in the dialog box are invisible, leaving only one visible schema at a time. At the bottom of the dialog is a save template button, which allows the user the choice of where to

store the template.

Implementation Details and Results

We could not immediately decide between using a website approach or JavaScript dialog approach to making the Schema Filler. It worked out well for us to try both approaches at the same time. The JavaScript dialog turned out to be the easier and more efficient approach to designing the Schema Filler in the end. We were having problems getting any website to automatically appear in the Bridge browser window. The JavaScript dialog, however, seamlessly integrates into our existing Schema Chooser dialog.

One of the problems we had with the XmlNode object was that any case sensitive attribute values come out as lower case. Because the XMP files are so touchy, it was necessary for us to have case sensitive tags. The problem with the XmlNode was very simply fixed, but since we did not want to overwrite the XmlNode object, we decided to basically copy and past, changing one line of the code. This new object is called XmlNodeCS.

We decided to write an object for writing to XMP. This has a method to start writing the XMP file, which writes the header, as well as a finish writing method, which writes the ending. The main body is written with a function that takes in a XmlNodeCS object. It is required that the XmlNodeCS structure has properly named tags for XMP, for instance, the tags must have the schema name appended with a colon to the tag name (i.e. Iptc4xmpCore:CiEmailWork). This was a quick approach, but it works fairly well.

The main requirement was that the application writes to an XMP file that Adobe Bridge could then embed to any pictures. We had to do a lot of work to get the tags just right in order for the XMP file to embed. To create a template in Adobe Bridge, you go to the File Info Panels, fill in information, and create the template. We did this and opened up the XMP file that the process generated. While you can create an XMP file that will embed information, if you do not follow Adobe Bridges exact structure for the standard schemas, then the XMP file will embed to the file, but will not place the information back into the File Info Panels.

Conclusions and Future Directions

In conclusion, this program has the possibility of affecting many photographers. It will help them to organize their pictures before they go into the field to do a photo shoot. This will increase their productivity and help to organize the pictures downloaded to their computer.

In the future, teams might want to figure out a way for users to easily obtain

certain predefined values, such as IPTC scene values. Another future development would be the ability to make a custom schema by adding information into the schema file. The schemas file will need to be updated with any schemas that Adobe declares as a standard schema.