Xaraya
Request for Comments: 0026
Category: Standards Track

P Fletcher
Stutchbury
November 2002

BlockLayout for Content

Status of this Memo

Copyright Notice

Abstract

This rfc describes the concept of allowing content to be parsed as a BlockLayout template.

It attempts to address the scope and implications of the changes required to implement this proposal.

Table of Contents

1. Introduction

This RFC is a first draft proposing the benefits and flexibility of parsing Xaraya content for BlockLayout tags. Please see the Appendix for a definition of terms used in the context of this document.

2. Current Implementation

The rendering of current Xaraya content is currently defined by one or more template files. Although the content can vary, the format in which it is presented is determined by one or more template files which, by their very nature, are static. BlockLayout provides an enormous amount of flexibility for implementors to create a unique website for a client. Template files will usually be created by the implementor for a client site and will not normally be modified by admins or editors.

3. BlockLayout Rendering for Content

This RFC proposes that, in addition to providing the rendering for Master, Slave and Internal blocks, BlockLayout should also be able to provide rendering for Content. It is adding a lower level of granularity to BlockLayout. This means that specifed content would be parsed as if it were a Template file. All BlockLayout tags would be valid.

For the purposes of this document, content is defined as any data stored in the database for the purpose of display, usually on a web site but could be published by any other means.

By using the BlockLayout DTD (or XML schema), Xaraya will define a standard structure for all dynamic content. Module developers will not have to re-invent the wheel, but will have a clearly documented, flexible and proven framework to build their data around.

Authorized admins, editors and even authors would be able to create content that is rendered in a truly unique fashion, within the defined boundaries of the static template files.

Third party modules, such as WYSIWYG editors or FormExpress :) would use the BlockLayout DTD to structure the storage and presentation of their content.

Changes required to the BlockLayout interfaces would greatly ease the creation of a database backed template editor in the future.

4. Security

In a CMS which accepts contributions from insecure (public) sources, one would not want all contributors to be able to include BlockLevel tags in their text which might reveal sensitive information or destroy the formatting of a page. This would have to be addressed at the data entry stage, preferably by providing a core function/method that module developers would call before committing content to the database.

The granularity of allowable BlockLayout tags could be developed over time, with a simple boolean value to begin with (i.e. all or nothing) and then extending the current 'Allowable HTML tags' functionality to include BlockLayout tags.

As an aside, it would be useful for implementors and admins to be able to specify custom 'Allowable Tags' and to be able to define wildcard matches to be permitted or denied.

On the display side, only content flagged for parsing by BlockLayout would be processed.

5. Performance

Assuming that BlockLayout has already been designed for performance, this proposal should not significantly impact page rendering speed.

Not all content would need to be parsed. A flag or code placed against a content column would instruct a module to submit the content to BlockLayout before rendering. Alternativley, an xml declaration could be made at the head of the content.

The same cacheing principals would apply - except that the md5 checksum would be applied to the content string rather than the template file. It would be beneficial (although initially not essential) for the module to request that BlockLayout does not cache the results. The implementation of cacheing rules for content could warrent further investigation, especially when combined with widget tags.

6. Solution Proposal

This section discusses how the BlockLayout for Content proposal could be implemented.

6.1 BlockLayout Interface Changes

The following public interfaces to BlockLayout are:

6.1.1 xarTplString()

Due to it's similar naming convention to xarTplFile(), this function appears to do what this proposal requires, but the two functions do not behave the same.

xarTplFile() will accept a BlockLayout Template file name; if that file has not already been compiled, then the blCompiler will be invoked and a cache file created.

xarTplString(), however will only accept the equivalent of the contents of a BlockLayout cache file i.e. valid PHP code, not BlockLayout code.

For consistency, the functionality offered by xarTplString() should be the same as that offered by xarTplFile i.e. the input should the an uncompiled BlockLayout Template. An appropriately named new function should be created to implement the current xarTplString() functionality e.g. xarTplEval()

7. Code that will need to be written

7.1 New Functions

7.1.1 xarTpl__executeFromString()

This new private function will implement cacheing for string based template parsing. It is loosely based around xarTpl__executeFromFile(). In fact, there is much commonality that should be brought together into shared functions.

```
          function xarTpl__executeFromString($sourceString, $tplData,
$cache_flag=true)
{
    global $xarTpl_cacheTemplates;

    $needCompilation = true;

    if ($xarTpl_cacheTemplates && $cache_flag) {
        $varDir = xarCoreGetVarDirPath();
        $cacheKey = md5($sourceString);
        $cachedFileName = $varDir . '/cache/templates/' . $cacheKey . '.php';
        //
        //Need to check if source string has changed - is md5 distinct enough to:
        //a) Change with minor amendments to sourceString? or
        //b) Not conflict with other cached files?
        //
        if (file_exists($cachedFileName) ) {
            $needCompilation = false;
        }
    }

    //xarLogVariable('needCompilation', $needCompilation, XARLOG_LEVEL_ERROR);
    if ($needCompilation) {
        $blCompiler = xarTpl__getCompilerInstance();
        $templateCode = $blCompiler->compile($sourceString);
        if (!isset($templateCode)) {
            return; // exception! throw back
        }
        if ($xarTpl_cacheTemplates && $cache_flag) {
            $fd = fopen($cachedFileName, 'w');
            fwrite($fd, $templateCode);
            fclose($fd);
        } else {
            return xarTpl__execute($templateCode, $tplData);
        }
    }
        $tplData['_bl_data'] = $tplData;
    // $__tplData should be an array (-even-if- it only has one value in it),
    // if it's not throw an exception.
    if (is_array($tplData)) {
        extract($tplData, EXTR_OVERWRITE);
    } else {
        $msg = 'Incorrect format for tplData, it must be an associative array of
arguments';
        xarExceptionSet(XAR_SYSTEM_EXCEPTION, 'BAD_PARAM',
                       new SystemException($msg));
        return;
    }

    // Start output buffering
    ob_start();

    // Load cached template file
    if (xarCoreIsDebuggerActive()) {
        $res = include $cachedFileName;
    } else {
        // Suppress error report when debugger is not active to prevent
        // that the var dir hash key could be stolen
        $res = @include $cachedFileName;
    }

    // Fetch output and clean buffer
    $output = ob_get_contents();
    ob_end_clean();

    // Return output
    return $output;
}
```

7.1.2 xarTplEval()

The public function xarTplEval() will simply replace the current functionality of xarTplString().

```
function xarTplString($templateCode, $tplData) {
          return xarTpl__execute($templateCode, $tplData);
          }
```

7.2 Modified Functions

7.2.1 xarTplString()

This will become the public interface to BlockLayout for Content. The current version of this function does not appear to be called from anywhere (at the moment), so changes should have no impact on existing code.

```
function xarTplString($templateCode, $tplData, $cache_flag) {
          return xarTpl__executeFromString($templateCode, $tplData, $cache_flag);
          }
```

8. Open Issues

This section lists some of the things I might not yet have a handle on or that may need to be investigated by someone who has a better understanding of the core Xaraya code

8.1 md5

md5 is used against the template source string to provide a file name for the cache. Is the result of md5 distinct enough to:

- Change with minor amendments to sourceString? or
- Not conflict with other cached files?

8.2 Garbage Collection

What requirements are there for garbage collection? i.e. removing stale content cache files.

9. Revision history

Version 1.0 First Draft.

10 Reference title

Author's Address

Philip Fletcher
Stutchbury
EMail: philip@stutchbury.com
URI: http://www.stutchbury.com

A. Definitions of the the Players

This section lists and describes the players or roles involved in developing and running a Content Management System

- Developer: Creates and maintains core and module code for general public release. May also create themes which are made available to others.
- Implementor: Installs and customizes Xaraya for clients. The customizations will be client specific such as modifying the BlockLayout templates to alter the look and feel of the site. Changes to core or module code will not always be rolled back into the Xaraya core.
- Admin: Looks after the day to day maintenance of an implemented site. Often a senior, technically competent user.
- Editor: Creates and approves articles and other content for publishing. Not necesarily technically minded.
- Author: A user who submits articles or other content for publication on a regular basis and would be trusted to maintain or amend existing content that they 'own'.
- Contributor or User: A visitor to the site. May or may not be logged in, may occasionally submit content, but will not take any further part in it's publication.

Intellectual Property Statement

Acknowledgement

Funding for the RFC Editor function is provided by the DDF