

Modification Markup Version 2.2

3D7 Software

This document describes the modification markup that can be used to mark changes and other features in code. It uses an XML-style format that can be used to find the editing tags quickly. It also allows them to be processed a mechanical fashion, such as with a script. Modification markup works best on code that is not version-controlled, but it is almost just as useful on versioned code. It also provides all of the necessary features to be used in situations where more than one person works on a particular piece of code. It can also be used to set reminders.

Contents

Modification Markup Version 2.2.....	1
Contents.....	1
Revision History.....	1
Attributes.....	1
Tags.....	3
File System Usage Differences.....	6
XML Namespace.....	7
Locating Tags.....	7
Modification Markup Quick Reference.....	9

Revision History

Version	Date	Author	Comments
1.0	2004-05-10	Brian Draeger, Brad Tetzlaff	The basic ideas of Modification Markup were formed when code comments began to take the form of "MODIFY: ..."
2.0	2004-06-04	Brad Tetzlaff	The first formalized XML-based version.
2.1	2005-12-26	Brad Tetzlaff	Adds better support for synchronization and time/version range specification.
2.2	2006-08-20	Brad Tetzlaff	Adds support for exact specification of what piece of code a markup tag refers, the <code>src</code> and <code>if</code> attributes, and file system definition.

Attributes

The attributes are always optional except for the identifier attribute (`i`) when used with the `sync` tag and in a few other circumstances. They should always appear in the same order that they appear here when more than one is used. Single letter attributes have been used to keep the amount of typing as short as possible.

`i` ("Identifier") - Sets the identifier that this tag uses to relate to other modification markup tags. Its primary use is for synchronization and code movement marking. The identifier is case sensitive.
Example: `i="Snippet1"`

`a` ("Author's Name") - Note the author who wrote the editing tag originally.
Example: `a="Brad"`

`ra` ("Response Author Name") - Designate who made the change or who responded to the request. It is not relevant for non-edit tags.
Example: `ra="Mark"`

r=" (Recipient's Name) " - Direct the edit comment to a particular developer or group of developers (use commas). This can be very useful if someone is reviewing code.
Example: **r="Bob, Mark"**

t=" (Timestamp) " - Record when the edit comment was originally made. The time should be formatted like the **dateTime** format of XML (<http://www.w3.org/TR/xmlschema-2/>), "yyyy-MM-ddTHH:mm:ss±TT:tt". TT:tt is the time zone offset from GMT (optional) and HH is a 24-hour clock value.
Examples:

```
t="2005-06-04T13:05:12-06:00"
t="2005-06-04T13:05"
t="2005-06-04"
```

st=" (Start Date) " - Mark a start date for the soonest time that the modification should be made. The time should be formatted like the **dateTime** format of XML schema (see attribute **t**).
Example: **st="2005-11-14"**

et=" (End Date) " - Mark the time when a change was actually made. The time should be formatted like the **dateTime** format of XML schema (see attribute **t**).
Example: **et="2005-11-24T07:30"**

due=" (Due Date) " - Mark a due date for a change. The time should be formatted like the **dateTime** format of XML schema (see attribute **t**). It is not relevant for non-edit tags.
Example: **due="2005-11-19"**

p=" (Priority) " - Give the tag a priority level. Levels are "" for very low, "!", for low, "!!", for medium "!!!" for high and "!!!!" for very high, or a long stream of exclamations to get any reader's attention.
Example: **p="!!!"**

v=" (Version) " - Marks the version of the code relevant to the comment.
Example: **v="Release 1.4"**

sv=" (Start Version) " - Marks the version of the code when the modification should be made.
Example: **sv="3.0"**

ev=" (End Version) " - Marks the version of the code after modification.
Example: **ev="3.1"**

title=" (Title) " - Provide a title for the comment. This attribute is rarely used and is most useful for multi-line notes. Example: **title="How This Code Works"**

x=" (What the Tag Refers to Exactly) " - Some modification markup tags may not refer to the whole next line of code. If the tag only refers to a part of the code, either describe the code here or place that code in this attribute so that the focus of the modification markup tag is understood clearly.
Example: **x="fooBar"** (where variable **fooBar** is in the referenced code)

sx=" (Start Expression) " - Indicates what the code looks like before the modification.
Example: **sx="oldValue"**

ex=" (End Expression) " - Indicates what the code looks like after the modification.
Example: **ex="newValue"**

src=" (Address) " - An address, code reference, file, or a comma-delimited list of any of these related to the tag.

Example: `src="http://www.3d7software.org/compsci/modmarkup.php"`

`if=" (Condition) "` - Describes the condition that must apply for the change to be applicable.

Example: `if="org.xyzcorp.FileBrowser.GetFiles() includes this code."`

`c=" (Description Here) "` - Describe the edit comment in detail.

Example: `c="This function does not work."`

Tags

The tags allow the code to be mark in various ways. The comments that start with "ED" all note some kind of modification. The other tags allow the code to be marked in special ways. The following are true of the tags:

1. Some tags come in three versions: imperative, past tense, and negative. The imperative version is used when something should be done but has not yet been done. The past tense version marks that a change was completed. The negative version tells that a change should not be done.
2. All of the tags are in uppercase letters to differentiate them from the actual code. This is especially useful when editing HTML, XML, or ColdFusion. It can prevent any chance of a naming conflict in XML since most tags are written in lowercase.
3. The tags should be used as XML, so they should be terminated with a forward slash if they have no content. All tags may be used with content or without content.
4. If an editing tag appears without content, then it matches with the next statement of code.
5. All of the attributes mentioned may be used in any of the tags. Some apply better than others.

The tags are as follows (the attribute name is followed by a list of applicable attributes where those in square brackets are optional):

```
<EDIT [i, a, r, t, st, due, p, v, sv, ev, title, x, sx, ex, if, c] />
<EDITED [i, a, ra, r, t, st, et, due, p, v, sv, ev, title, x, sx, ex, if, c] />
```

```
<EDITNO [i, a, r, t, st, et, p, v, sv, ev, title, x, sx, ex, if, c] />
```

- Marks a general modification. The code should be edited, was edited, or should not be edited. (Note that "EDITED" was chosen since "MODIFIED" changes the 'Y' to an 'I' and therefore some search expressions would need to be more complex.) Examples:

```
// <EDIT sv="2.5" sx="128" ex="256" />
var = 128;

// <EDITED ra="Anton" et="2005-10-16T14:25"
// c="I worked on this code more.">
...
// </EDITED>
```

```
<EDADD [i, a, r, t, st, due, p, v, sv, ev, title, x, src, if, c] />
<EDADDED [i, a, ra, r, t, st, et, due, p, v, sv, ev, title, x, src, if, c] />
<EDADDNO [i, a, r, t, st, et, p, v, sv, ev, title, x, src, if, c] />
```

- Marks that code should be added, was added, or should not be added. Use an identifier attribute (i) to mark it as a place to move or copy code. Examples:

```
// <EDADD a="John" r="Ron" t="2005-06-07" due="2005-06-18" c="I'm
// going to need to have that sorting function here before I can start
// working on the file output." />
```

```
' <EDADDED />
Dim newVariable As String

<!-- <EDADD i="ServiceCode" src="../../service1.js?activateService"
c="Put this function here instead." /> -->
...
```

```
<EDREMOVED [i, a, r, t, st, due, p, v, sv, ev, title, x, if, c] />
<EDREMOVED [i, a, ra, r, t, st, et, due, p, v, sv, ev, title, x, if, c] />
<EDREMOVED [i, a, r, t, st, et, p, v, sv, ev, title, x, if, c] />
- Marks that code should be removed, was removed, or should not be removed. Example:
```

```
<!-- <EDREMOVED if="The alternate method does not work." /> --->
<cfset random=Rand()>
...
<!-- </EDREMOVED --->
```

```
<EDIMPROVED [i, a, r, t, st, due, p, v, sv, ev, title, x, sx, ex, if, c] />
<EDIMPROVED [i, a, ra, r, t, st, et, due, p, v, sv, ev, title, x, sx, ex, if,
c] />
<EDIMPROVED [i, a, r, t, st, et, p, v, sv, ev, title, x, sx, ex, if, c] />
- Marks that the code should be improved, was improved, or should not be improved. It can be used to
remind a developer that the code was written simply for it to work, but a more efficient solution is likely to
be available. Example:
```

```
// <EDIMPROVED v="Debug" c="Don't work on this until later since we
// have too many other tasks to work on." />

// <EDIMPROVE a="Brad" c="I put this together quickly at the end of the
// day." />
```

```
<EDRENAMED [i, a, r, t, st, due, p, v, sv, ev, title, x, sx, ex, if, c] />
<EDRENAMED [i, a, ra, r, t, st, et, due, p, v, sv, ev, title, x, sx, ex, if,
c] />
<EDRENAMED [i, a, r, t, st, et, p, v, sv, ev, title, x, sx, ex, if, c] />
- Marks that a particular name should be changed, was changed, or should not be changed. Example:
```

```
// <EDRENAMED st="2006-01-04" ex="greatName" />
int badName = 54;
...
// <EDRENAMED st="2006-01-04" et="2006-01-09" sx="badName" />
int greatName = 54;
```

```
<EDFIX [i, a, r, t, st, due, p, v, sv, ev, title, x, sx, ex, if, c] />
<EDFIXED [i, a, ra, r, t, st, et, due, p, v, sv, ev, title, x, sx, ex, if, c]
/>
<EDFIXNO [i, a, r, t, st, et, p, v, sv, ev, title, x, sx, ex, if, c] />
- Marks that the code should be fixed, was fixed, or should not be fixed. It also represents that there is a
bug in the code. Example:
```

```
<!-- <EDFIX --->
<cfif 5 DIV 0 EQ 8>
```

```

    <cfquery name="fooBar" datasource="NET1">
    ...
    </cfquery>
</cfif>
<!--- </EDFIX> ---->

```

```

<TEST [i, a, r, t, st, due, p, v, sv, ev, title, x, if, c] />
<TESTED [i, a, ra, r, t, st, et, due, p, v, sv, ev, title, x, if, c] />
<TESTNO [i, a, r, t, st, et, p, v, sv, ev, title, x, if, c] />
- Marks that the code needs testing, was tested, or should not be tested. Examples:

```

```

<!--- <TEST r="Mark" due="2005-07-01"> ---->
<cfloop index="i" from="0" to="30">
    <cfhttp ...>
    ...
    </cfhttp>
</cfloop>
<!--- </TEST> ---->

// <TESTED c="This code really works well." />

```

```

<SYNC i [a, ra, r, t, st, et, p, v, sv, ev, title, x, src, if, c] />
<SYNCNO i [a, r, t, st, et, p, v, sv, ev, title, x, src, if, c] />
- Indicates that synchronization must be made or should not be made for a particular block of code.
Every synchronization tag should have an identifier attribute (i) and a matching SYNC tag with the same
identifier. The code at each of these locations should be kept exactly the same. This can be very useful
for cases where a piece of code is must be shared, but is best manually inlined for efficiency or packaging
reasons. Example:

```

```

// File: map.php
// <SYNC i="Item0Title" x="Item 0" src="list.php">
    if ($itemPath->title === 'Item 0') {
        $isValid = false;
    }
// </SYNC>

// File: list.php
// <SYNC i="Item0Title" x="Item 0" src="map.php">
    if ($itemPath->title === 'Item 0') {
        $isValid = false;
    }
// </SYNC>

```

```

<DESC [i, a, r, t, st, et, p, v, sv, ev, title, x, if, c] (at least one
attribute required) />
- Describes the code using the attributes. For instance, the a attribute marks the author and v marks the
code version. If an identifier attribute (i) is given and it has a matching EDADD tag, then this means that it
code should be copied. Example:

```

```

# <DESC a="Brad" v="XYZ Corp Only" p="!">
while (x > y) {
    ...
    x--;
}

```

```
# </DESC>
```

```
<REF [i, a, r, t, st, et, p, v, sv, ev, title, x, src, if, c] (either i, src,
or c is required) />
```

- Informs the code reader of a reference that could be used to learn more about the code. If an identifier attribute (*i*) is given, then it can be used to reference code that is related. Use the *src* attribute to specify a specific address to which to refer. Examples:

```
<!--- <REF a="Brad" r="Joe Smith" c="Talk with Tina about this code.
She told me that you should have this some other way." /> --->
```

```
<!--- <REF src="Xyz.Corp.IO.CommPort.Open(BaudRate)" /> --->
```

```
// <REF i="FileIO">
FileStream fs = new FileStream(path);
fs.open(true);
startProcess(fs, x);
// </REF>
```

```
... // A long segment of code here.
```

```
// <REF i="FileIO" c="Be sure to update this code if you change
// the code above.">
fs.close();
endProcess(x);
// </REF>
```

```
<NOTE [i, a, r, t, st, et, p, v, sv, ev, title, x, if] c (optional if there
is a body) />
```

- Allows notes to be added and found easily. The note should be the content of the tag if it has content. One advantage to using notes like this is that they can be found easily by a script. It is not recommended for common comments though. Example:

```
/*
<NOTE title="A Long Comment">
A note is used for longer comments. It can also be used to add notes in
situations where a script automatically gathers all of the tags.
</NOTE>
*/
```

```
// <NOTE p="!!!!" c="The content type must be 'text/xml'." />
```

Other tags could be added for special purposes in some projects.

File System Usage Differences

During development there may be some usefulness in annotating a particular file or directory using Modification Markup. Some file systems in use do not allow the common XML characters of '<', '>', '/', and '"' to be used in a file name. Therefore, the following guidelines should be made when using Modification Markup in a file system that does not have any formal support for file or folder annotation.

1. Any Modification Markup tags should be written as folder names. A new folder should be created each time a new Modification Markup tag is added.

2. Since the XML characters of '<' and '>', are normally illegal in a file name, a Modification Markup tag should be written with a leading '##' symbol followed by the name of the tag. Attributes should be separated by whitespace. No terminator '/' should be written and all tags are assumed to be relevant to the current directory unless an **x** attribute mentions a specific file or files.
3. Since the '"' character is normally illegal in file names, this a single quote mark, '' should be used instead. Any other illegal file name characters that appear in the attribute values should be written as XML macros.
4. File paths are likely to be mentioned in the attribute values. The directory separator character is definitely illegal in a file name, so a '%' may be used alternatively to maintain readability.

Examples:

```
C:\Projects\Team Project 1\##DESC a='Zhao, Jodi' due='2007-06-01'
```

```
C:\Projects\Team Project 1\Documentation\##EDRENAME sx='DesignSpec.odt' ex='Project 1&#39;s
Design Specification 1.0.odt'
(Where "DesignSpec.odt" is file in the Documentation folder.)
```

```
/usr/test2000/##EDADD i='t1-t2' a='Ron' r='Jack' c='Move the files from %usr%test1000 to here.'
```

XML Namespace

Modification Markup is normally not used with any namespace specification. If, however, there is a need to differentiate the markup from other tags, an XML namespace prefix of "mm" set to the namespace of "http://www.3d7software.org/modmarkup/v2-2" may be used. Example: `<mm:EDADD ...`

Locating Tags

The best part of the modification markup tags is that every tag can be found without difficulty and according to its attributes. A regular find operation is capable of finding the tags, while a regular expression search is required for finding attributes simply because there is no other way. The ability to insert modification tags anywhere necessary allows a developer to relate the comments directly to the code. It can also reduce errors since necessary modification reminders can be entered instantly in relation to the code rather than having to write them down somewhere else.

Automated Processing

If you plan to use a script to mechanically find Modification Markup within code, it is recommended that you use an ampersand (&) before a tag to make it easier for the automated processor to find the Modification Markup tags. This is not likely to be needed in most cases, but it could be important if you are writing XHTML code that includes uppercase tags with similar names (you could also choose to use namespaces then.) Examples:

```
<!-- &<EDIT c="Edit this code." /> -->

/* &<SYNC i="CodePart1"> */
...
// &<SYNC />
```

Some example search strings follow:

Simple Search Strings

"<ED" - Find all editing tags.

"<EDREMOVED" - Find all completed code removals.

"<TEST " - Find all of the tests that must be done.

"<EDFIX" - Find all of the bugs that are known.

"<EDFIX " - Find all of the bugs that still need to be fixed.

"<SYNC i="SyncName"" - Find all of the "SyncName" code blocks that must be kept exactly the same.

Regular Expression Search Strings

(The regular expressions shown here are according to the .NET Framework format - <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconRegularExpressionsLanguageElements.asp?frame=true>)

"\<ED[A-Z]+ED " - Find all completed modifications.

"\<ED[A-Z]+(E|T|X|DD) " - Find all modifications that need to be completed.

"\<DESC.* v="1\0"" - Find all code that is marked as version 1.0.

"\<[A-Z].+ r=".*John.*"" - Find any tag marked for John to know about.

"\<ED.+ due="2005-05.*"" - Find any modification due in May of 2005.

"\<[A-Z].+v="1\.[345].*"" - Find any modification related to version 1.3, 1.4, or 1.5.

"\<[A-Z].+ p="!!+"" - Find all of the tags that are at least a medium priority tags.

"\<ED.+x="xmlScript"" - Find modification referring to the variable `xmlScript`.

"\<[A-Z].+c=".*http://www.w3c.org.*"" - Find any time <http://www.w3c.org> was mentioned as a source or in a comment.

(Note that these examples assume no & was before tag names and that tags were kept on one line. These examples may return erroneous results in extremely rare cases.)

Modification Markup Quick Reference

A summary of Modification Markup version 2.2:

Tags	Attributes
<code><EDIT -ED -NO /></code> - Edit code	<code>i</code> - Identifier for relation of tags
<code><EDADD -ED -NO /></code> - Add code	<code>a</code> - Original author's name
<code><EDREMOVE -D -NO /></code> - Remove code	<code>ra</code> - Responder (who made the change)
<code><EDFIX -ED -NO /></code> - Fix code/bug	<code>r</code> - Recipient's name
<code><EDIMPROVE -D -NO /></code> - Improve code	<code>t</code> - Original timestamp
<code><EDRENAME -D -NO /></code> - Rename item	<code>st</code> - Start date
<code><TEST -ED -NO /></code> - Test code	<code>et</code> - Completion date
<code><SYNC -NO /></code> - Keep code synchronized	<code>due</code> - Due date
<code><DESC /></code> - Describe code	<code>p</code> - Priority as a set of '!'
<code><REF /></code> - Make a reference	<code>v</code> - Related version
<code><NOTE /></code> - Make a note	<code>sv</code> - Initial version
	<code>ev</code> - Completion version
	<code>title</code> - Title of note or code
	<code>x</code> - What the tag refers to exactly
	<code>sx</code> - Unmodified code expression
	<code>ex</code> - Modified code expression
	<code>src</code> - The address of a source of info.
	<code>if</code> - Applicable only under conditions
	<code>c</code> - Comment/Description

- `-ED` is used to mark that a change was made.
- `-NO` is used to mark that this change should not be made.
- No suffix (present tense) means that the change should be made.
- Attributes should appear in the order listed above.
- A date is formatted like `yyyy-MM-dd`.
- A date/time is formatted like `yyyy-MM-ddTHH:mm:ss±TT:tt` where `HH` is a 24-hour value. `TT:tt` is the time zone offset from GMT (optional).
- Use the identifier attribute (`i`) with the `SYNC` tag to identify all of the instances of the code. Use it with a `DESC` tag and an `EDADD` tag to copy code, or `EDREMOVE` and `EDADD` to move code.
- Use a `&` before every Modification Markup tag if you want to ensure that they can be found very easily without error.
- If using Modification Markup in a file system each tag is a folder and tag names start with `##`.

General Example:

```
// <EDITED a="Rob" r="Mike" t="2006-10-20T15:56" p="!!!" ev="1.3"
// c="I changed a few items in this code for v 1.3,
// you should take a look.">
...
// </EDITED>
```