MUD eXtension Protocol (MXP)

The MUD eXtension Protocol (MXP) is an open specification for enhancing the communication between MUD servers and clients. To contribute to this specification, email zugg@zuggsoft.com or participate in the Developer's Forum on www.zuggsoft.com.

- MXP specification
- List of MUDs with MXP support
- List of MUD Clients with MXP support
  - zMUD Implementation notes
  - MUSHclient Implementation notes

MXP Specification

Version 1.0 (12-Mar-03)

Contents

Introduction

MUD Servers communicate with MUD Clients via the Telnet Protocol. While Telnet is the basis of most Internet protocols (FTP, HTML, SMTP, etc), most of these protocols enhance Telnet with their own higher-level protocol in order to provide more specific and directed features. For example, FTP adds a command language for transferring files via the Telnet socket connection.

Traditionally, MUDs have used raw Telnet. MUD sessions have been enhanced using standard terminal protocols, such as ANSI and VT100 to add color and cursor control. However, there is no higher-level command protocol used by MUDs to enhance the game session. The MUD eXtension Protocol (MXP) attempts to fill this gap by providing a markup language based loosely upon HTML and XML that allows better communication between the MUD Server and Client.

The design goals for MXP are:

1. MXP will be based upon existing standards when possible.
2. MXP will be small and efficient since MUDs send a great deal of text to the client already.
3. MXP will be easy to implement for both the MUD Server and Clients.
4. The MXP specification will be public-domain.

NOTE: The copyright for the MXP protocol has been removed by Zugg Software as of MXP version 0.1. Zugg Software will still lead the development of MXP, but any MUD client or server is welcome to use and extend this specification as desired. Zugg Software will coordinate the documentation of MXP in the Developer's Forum at the web site: www.zuggsoft.com. Send all proposed changes to zugg@zuggsoft.com to get them incorporated into this spec.

Overview of MXP

MXP is based loosely on the HTML and XML standards supported by modern web browsers. It is only "loosely" based upon these standards because MUDs require dynamic, streaming text, rather than the page-oriented view of web browsers. Also, the existing standards are needlessly verbose for use on a MUD where text efficiency is important.

In addition, there are additional security concerns to worry about with MXP. While support for HTML tags within a line is desired, players on a MUD can exploit this power and cause problems. Certain HTML functions need to be restricted so that MUD players cannot abuse them. For example, while it might be desirable to allow players to change the font or color of their chat messages, you would not want to allow them to display a large graphics image, or execute script commands on the client of other users. MXP handles this by grouping the various tags and commands into secure and open categories, and preventing the secure tags from being used by MUD players.

Two additional Client/Server protocols are not discussed in this document. The MUD Sound Protocol (MSP) is a separate specification, and while MXP provides compatibility with MSP, it does not add any additional sound support of its own.

MXP Line Tags

The core of MXP involves tagging text lines sent by the MUD. Each MXP command is marked as either "Open" or "Secure". Lines sent from the MUD are also marked as "Open" or "Secure". On an "Open" MUD Line, only "Open" MXP commands are allowed. On a "Secure" MUD line, both "Open" and "Secure" MXP tags are allowed. Additional "open" tags can also be used to group lines from the MUD into channels that the user can easily control on the client.

MUD Server Implementation Note: Be very careful when sending Secure lines from the MUD. Be absolutely sure that MUD players cannot control the output of a secure line. If a MUD player is able to send a secure MXP command, he will be able to cause great damage to other MUD players using MXP.

To ensure that tags are difficult to send by MUD players, an escape sequence, similar to ANSI or VT100 is used:

```
<ESC> [ # z
```

where # is the tag number to use. Tag numbers are defined as:

0 open line
only MXP commands in the "open" category are allowed. When a newline is received from the MUD, the mode reverts back to the Default mode. OPEN MODE starts as the Default mode until changes with one of the "lock mode" tags listed below.

1 secure line (until next newline)
all tags and commands in MXP are allowed within the line. When a newline is received from the MUD, the mode reverts back to the Default mode.

2 locked line (until next newline)
no MXP or HTML commands are allowed in the line. The line is not parsed for any tags at all. This is useful for "verbatim" text output from the MUD. When a newline is received from the MUD, the mode reverts back to the Default mode.

The following additional modes were added to the v0.4 MXP spec:

3 reset
close all open tags. Set mode to Open. Set text color and properties to default.

4 temp secure mode
set secure mode for the next tag only. Must be immediately followed by a < character to start a tag. Remember to set secure mode when closing the tag also.

5 lock open mode
set open mode. Mode remains in effect until changed. OPEN mode becomes the new default mode.

6 lock secure mode
set secure mode. Mode remains in effect until changed. Secure mode becomes the new default mode.

7 lock locked mode
set locked mode. Mode remains in effect until changed. Locked mode becomes the new default mode.

The following modes are client-specific and were implemented in the v0.3 MXP spec by zMUD:

10 room name
the line is parsed as the name of a room. Only used by MUDs that only support tagging for the automapper and do not support the full MXP tag set.

11 room description
the line is parsed as a description of a room. Only used by MUDs that only support tagging for the automapper and do not support the full MXP tag set.

12 room exits
the line is parsed as an exit line for a room. Only used by MUDs that only support tagging for the automapper and do not support the full MXP tag set.

19 welcome text
This text is sent from the MUD at the beginning of a session to welcome the user to the MUD. This text is not displayed by the client if a <RELOCATE> command has been used. See the section on Multi-Server MUDs for more details. Same as the <WELCOME> MXP tag.

20-99 user-defined

the line is tagged for a MUD-specific purpose. The tag must have been previously defined using an MXP definition command to tell the client how to handle the line. A common use for these is to tag various chat channels to allow client-side filtering or redirection of the text.

Note that the tag numbers from 10-19 are reserved for automapper usage. MUDs can easily support the automapper by simply tagging lines sent to the client using these codes. However, MUDs that fully support the MXP spec should mark the automapper information using the MXP tags described later rather than just tagging the line.

As with VT100 and ANSI sequences, the tag number is sent as decimal text. So, for example, to tag a line as "secure", the sequence: <ESC>[1z is sent from the MUD (where <ESC> is ascii character 27).

When the mode is changed from OPEN mode to any other mode, any unclosed OPEN tags (tags that were used while in open mode) are automatically closed. Also, when in OPEN mode, any unclosed OPEN tags are automatically closed when a newline is received from the MUD.

Note that secure tags are never automatically closed (this is a change from the 0.3 spec). Be sure to close your secure tags sent from the MUD, or use the Reset mode periodically.

The concept of the Default mode was added in the 1.0 version of the MXP spec to clarify how the locked tags work, and how the mode is changed when a newline is received.

MXP Reference

The core of MXP involves "Elements." Elements are like normal HTML tags. For example <B> is an element called "B". It causes text to be bolded. To turn off the bold, the </B> element is used. Each element has a corresponding closing element that starts with a /. The exception are Commands. Commands are elements that do not require a closing tag. For example, the element
 causes a line break. No closing tag is needed.

Borrowing a feature from XML, MXP allows you to define your own elements. This is the true power of MXP. By defining your own elements and giving them short names, you can cause complex output formatting in only a few short characters.

In addition to Elements, you can also define "Entities." Entities are like macro string replacements. For example, in HTML, the entity < indicates the < or less-than symbol. Since a normal < symbol is interpreted as the start of an element tag, you must use < to refer to a less-than symbol directly. Entities are accessed by putting a & character in front of the entity name, and terminating the name with a ; character. All of the standard HTML entities are available in MXP, including the &#nnn; entity to insert character /nnn into the text stream. Note that nnn values less than 32 are ignored.

Of course, like in XML, you can define your own entities in MXP. MUD-defined entities work much like server variables. For example, you could store the player's hit-points in an entity called &hp;

Here is a list of each MXP command, along with it's purpose and abbreviation:

Elements

<!ELEMENT> <!EL>
<!ELEMENT element-name [definition] [ATT=attribute-list] [TAG=tag] [FLAG=flags] [OPEN] [DELETE] [EMPTY]>

Used to define a new element (user-defined tag). The name of the tag is specified. To specify a Command element with empty content (such as a simple command tag), add the keyword EMPTY after the name of the tag. For example:

```
<!EL Command EMPTY>
```

specifies a tag called <Command> which does not require a closing </Command> since it has no content.

The 'Definition' is the optional macro expansion for this element. You can use a list of any text or other MXP commands in this definition. For example, to create a new element called "RED" which colors text red and makes it bold, you would define it as:

```
<!ELEMENT red '<COLOR red><B>'>
```

And then you could use it in your MUD output like this:

```
<red>This text is bold and red</red>
```

The 'Attribute-list' allows you to define arguments or attributes for your element. You can optionally specify the attributes using the <!ATTLIST> MXP command described later, or you can include them in the <!ELEMENT> definition for simplicity. To create an element that would allow you to change the text color, but would default to red, you would do:

```
<!ELEMENT boldtext '<COLOR &col;><B>' ATT='col=red'>
```

Then you could use it on the MUD like this:

```
<boldtext>This is bold red</boldtext>
<boldtext col=blue>This is bold blue text</boldtext>
<boldtext blue>This is also bold blue text</boldtext>
```

Note that you use an attribute in the element definition by referring to it as an entity. In this case, the temporary entity &col; refers to the color attribute that is passed. The 'col=red' defines the attribute, and gives the default value of 'red'. When the new element is used, the name of the attribute can be omitted if you give the arguments in the same order as the definition. That is how the third example is able to work.

The TAG=tag argument allows you to associate an element with a user-defined line tag. This is explained more in a later section.

The FLAG=flags argument allows you to assign an internal action to the element. This is explained more in a later section.

The OPEN argument sets the defined element to be an "Open" element. By default, elements are created as "Secure" elements.

The DELETE argument allows you to delete an element previously defined.

Security Note: Elements are marked as to whether they are created by the MUD or created interactively by the user. The user cannot override Secure elements sent from the MUD. The user is only allowed to override Open elements sent from the MUD.

Attributes

<!ATTLIST> <!AT>
<!ATTLIST element-name attribute-list>

As described in the previous section, this allows you to add attributes to elements. An attribute list specifies the name of the attribute and an optional default value. In MXP, attribute names can be eliminated from the Element if they are given in the same order that they are defined.

Implementation Note (MXP v0.1): Attribute lists were changed from the earlier MXP spec. The data type of the attributes are no longer specified, since the type is not important to the MUD server of client.

To define attributes, simply list the names of the attributes in the order that you want. To add an optional default, use the syntax =Default after the attribute name. For example the command:

```
<!ATTLIST boldtext 'color=red background=white flags'>
```

Specifies the attribute list for the 'boldtext' element. The first argument is called 'color' and has a default value of 'red'. The second argument is called 'background' and has a default value of 'white'. The third argument is called 'flags' and has no default value.

Note that you can specify the attribute list using the ATT='list' argument in the <!ELEMENT> definition. Because of this, there is little use for the <!ATTLIST> command except for modifying the argument list on-the-fly without changing the original definition.

For example, here is the full MSP Sound command as defined in MXP:

```
<!ELEMENT Sound EMPTY>
<!ATTLIST Sound FName V=100 L=1 P=50 T U>
```

The full usage of this element would normally be:

```
<SOUND FName="ouch.wav" V=50 L=2 P=80 T="combat" U="http://www.zuggsoft.com/sounds/">
```

But in MXP, the shorthand could be used:

```
<SOUND "ouch.wav" 50 2 80 "combat" "http://www.zuggsoft.com/sounds/">
```

You can omit some parameters using "" to specify the default, or mix attribute names in:

```
<SOUND "ouch.wav" "" 2 T="combat">
```

would be the same as

```
<SOUND FName="ouch.wav" V=100 L=2 T="combat">
```

Note the "" to specify the default for the V attribute. The P and U attributes also take their defaults since they are not specified at all. The T= overrides the parsing and tells MXP that the following value is for the T attribute instead of the P attribute that it was expecting at that position in the element.

Entities

<!ENTITY> <!EN>
<!ENTITY Name Value [DESC=description] [PRIVATE] [PUBLISH] [DELETE] [ADD] [REMOVE]>

Entities in MXP are used to store information from the MUD (MUD Variables). Once an entity is defined, you can reference it's value using the &Name; syntax. For example:

```
<!ENTITY Version "6.15">
The current version of zMUD is &version;
```

would display:

```
The current version of zMUD is 6.15
```

Tags can be included within entities. Unlike in XML, tags can be broken across entities within MXP as long as when all entities on a line from the MUD are expanded, all of the tags are properly closed (or MXP will close the dangling tags for you). So, the following is valid in MXP:

```
<!ENTITY Start "<em>">
<!ENTITY End "</em>">
&Start;This text is emphasized&End;
```

Also note that as in XML, entities are case sensitive, so &Start; is different than &start;. Unlike XML, entities can be used anywhere in MXP, even within other tags. However, if an external entity has the same name as an attribute, the attribute within a tag takes precedence.

All of the default HTML entities are defined in MXP.

To delete an entity, use the DELETE argument. Setting an entity to a empty value does not delete it.

PRIVATE entities cannot be queried by the MUD client. They are completely hidden.

PUBLISH entities can be used by the client to produce a list of MUD Server variables to be access by the player

The DESC description is used to give your entity a longer description

The ADD argument causes the Value to be added as a new item in a string list. So, it is appended to the existing value of the variable. String lists are values separated by the | character.

The REMOVE argument causes the Value to be removed from the existing string list.

Security Note: Entities are marked as to whether they are created by the MUD or created interactively by the user. The user cannot override Entites sent from the MUD. The user is only allowed to override user-defined entities.

<VAR> <V>
<VAR Name [DESC=description] [PRIVATE] [PUBLISH] [DELETE] [ADD] [REMOVE]>Value</VAR>

The <!ENTITY> tag allows the MUD server to set the value of a variable without displaying the value to the user. The <VAR> tag is just like the <!ENTITY> tag, except that the value of the variable is placed between the <VAR> and </VAR> tags, and this value is displayed to the user.

```
Hp: <VAR Hp>100</Var>
```

would display: "Hp: 100" to the user, and would set the entity hp to 100.

User-defined Line Tags

As mentioned near the beginning of this spec, a simple escape sequence can be used to tag a line. The tags 20-99 are defined by the MUD. The main purpose for user-defined line tags is to allow the user to customize how these lines are displayed. In particular, if the tag element is Open, the user can change the color, gag, or redirect the tagged line. This is useful for various chat channels on the MUD. Simply give them a name, and perhaps a default color, and make them open to allow the client user to gag them or redirect them to a different frame or window. Here is an example of an auction channel that can be controlled by the client user:

```
<!ELEMENT Auction '<FONT COLOR=red>' TAG=20 OPEN>
<ESC>[20zA nice shiny sword is being auctioned.
<Auction>Also, a gold ring is being auctioned.</Auction>
```

This also shows the two different ways the MUD server would then send text on this new auction channel. The use of the user-defined tag (<ESC>[20z) sends less text to the client, although the regular use of the tag name is easier to read and understand.

In both cases, the line is displayed to the user in red, by default. However, the user can change the attributes of this tag in the client, allowing them to change the color, gag the line, or redirect it to a different window.

Note that if the OPEN tag is left out, the user cannot change how the line is displayed.

<!TAG Index [WINDOWNAME=string] [FORE=color] [BACK=color] [GAG] [ENABLE] [DISABLE]>

Using the <!TAG> tag, the MUD can change the properties for a line tag.

INDEX is the tag number (20-99) to change.

WINDOWNAME specifies the name of a window to redirect the text to.

GAG indicates that the text should be gagged from the main MUD window.

FORE is the text color.

BACK is the background color of the text.

ENABLE and DISABLE can be used to turn this tag on or off.

For example, the tag defined above could also be manipulated like this:

```
<!ELEMENT Auction TAG=20>
<!TAG 20 Fore=red>
<ESC>[20zA nice shiny sword is being auctioned.
<Auction>Also, a gold ring is being auctioned.</Auction>
<!TAG 20 Fore=blue>
<ESC>[20zA nice shiny sword is being auctioned.
<Auction>Also, a gold ring is being auctioned.</Auction>
```

The first two lines would appear in red, and the second two lines would appear in blue.

Tag Properties

MUD Clients can define properties tags to be used by the server in !ELEMENT definitions. The syntax is FLAG='value'. The following special flags are defined in zMUD:

RoomName
The text for the element is parsed by the automapper as the name of a room.
RoomDesc
The text for the element is parsed by the automapper as the description of a room.
RoomExit
The text for the element is parsed by the automapper as exits for the room
RoomNum
The text for the element is parsed by the automapper as a room number
Prompt
The text for the element is parsed by as a MUD Prompt
Set
The text for the element is stored into the named local variable within the client. For example:

```
<!ELEMENT Hp FLAG="Set hp">
<Hp>100</Hp>
```

would set the @hp variable in the user's session to 100.

MXP Tags

Text Formatting

NOTE: Only the tags described in this section are OPEN tags. All other MXP tags are SECURE tags.

<B> <BOLD> <STRONG>
Render the text using a bold font. Any above tag name can be used for HTML compatibility, although the <B> tag is preferred for MUD usage.
Example: <B>this text is bold</B>
<I> <ITALIC> <EM>
Render the text using an italic font. Any above tag name can be used for HTML compatibility, although the <I> tag is preferred for MUD usage.
Example: <I>this text is italic</I>
<U> <UNDERLINE>
Underline the text. Any above tag name can be used for HTML compatibility, although the <U> tag is preferred for MUD usage.
Example: <U>this text is underlined</U>
<S> <STRIKEOUT>
Strike-out the text. Any above tag name can be used for HTML compatibility, although the <S> tag is preferred for MUD usage.

Example: <S>this text is crossed out</S>

<C> <COLOR FORE=foreground [BACK=background]>
Set the color of the text. If the background color is omitted, the current background color is used.
Example: <C red>This text is red</C>

<H> <HIGH>
Sets the color to a brighter version of the current color.

<FONT FACE=name [SIZE=size] [COLOR=foreground] [BACK=background]>
Change the font for the text. You can change the color at the same time instead of using a separate <COLOR> tag. Note that changing the font size can cause display problems on some clients.
Example: <FONT "Times New Roman">This text is in a fancy font</FONT>

Line Spacing

Also, unlike HTML, normal spacing still applies to the MUD output in MXP. So, anywhere that there is a newline in the MUD output, a new line appears on the client. Multiple spaces are not absorbed, but are displayed as sent by the MUD. Tab characters can still be used for tab stops, and ANSI and VT100 codes can still be used as normal. This provides a very rich output environment for the MUD which has the best of both worlds between standard Telnet text and HTML tags.

<NOBR>
To prevent a line break use the <NOBR> tag at the end of the line from the MUD. The next newline after the <NOBR> tag is ignored.

<P>
Mark a paragraph. A paragraph is a set of separate lines sent by the MUD which are recombined into a single line by the client. Basically, all newlines sent from the MUD between the <P> and the </P> are ignored. To force a line break, use

**
**
Line break. Forces a line break inside or outside of a paragraph. Note that
 is NOT parsed as a newline from the MUD as far as mode changes are concerned.

<SBR>
Soft line break. This is displayed as a space character in the client. However, it marks a preferred location on the line for a line-break to occur when word-wrapping the text in the client.

** **
As in HTML, the   entity can be used to indicate a non-breaking space character. Text will not be word-wrapped at this space by the client.

Links

<A href=URL [hint=text] [expire=name]>
The <A> tag works as it does in HTML to open a web page in the user's web browser. When the mouse hovers over the link, the optional "hint" text is displayed. If the hint text is not given, the URL of the link will be shown in the hint box. The Expire name is used for links that expire. See the <expire> tag for more details.

The http:// URL will open the user's external web browser to the specified web page. The telnet:// URL will open another window for the specified telnet address and port. The mailto: URL will open your email client allowing you to send email to the specified address.

Example: <A "http://www.zuggsoft.com>Click here for zMUD</A>

<SEND [href=command] [hint=text] [prompt] [expire=name]>
Send links allow you to initiate certain actions when a user clicks on linked text. On the web, clicking on a link results in opening a new web document. In MXP, links are more flexible, and normally will send commands back to the MUD rather than opening documents. For example, to send the command "buy bread" to the MUD, you would use the syntax:

```
<send>buy bread</send>
```

The MUD client would display "buy bread" as a link (underlined). If the user clicks on this link, the text "buy bread" is sent to the MUD.

You can make the text sent by the link and the text displayed to the user different using the HREF attribute for the <send> command. You can refer to the text being displayed using the entity value of &text; For example, if displaying a list of items to be purchased from the store when the user clicks on the item, you could send:

```
<send href="buy bread">bread</send>
<send href="buy water">water</send>
```

This would display the text "bread" and "water" and send the appropriate "buy" command when they are clicked. To minimize the text sent by the MUD in this case, it would be easier to set up your own Element for the store. For example:

```
<!ELEMENT Item '<send href="buy &text;">'>
<Item>bread</Item>
<Item>water</Item>
```

Would display the same list and still send the correct "buy" command to the MUD when the item is clicked on. In each case, the &text; entity is filled in with the text between the opening <Item> and closing </Item> tags.

Of course, you can omit the HREF= part since the first attribute of <send> is the command text. So, the real short form of this example would be:

```
<!EL Itm '<send "buy &text;">'>
<Itm>bread</Itm>
<Itm>water</Itm>
```

(MXP v0.2): You can change the mouse-over text of the link using the HINT="text" argument to the SEND tag.

<SEND> menus

You can create a popup menu of several commands by using a string list in the <SEND> command. Simply place a vertical bar | character between each command and each hint text. The hint text is used for the caption of the popup menu items. When a particular menu item is selected by the user, the command is sent to the MUD. If the Hint text contains one additional string, it is used as the mouse-over text. For example:

```
<send "command1|command2|command3" hint="click to see menu|Item 1|Item 2|Item 2">this is a menu link</SEND>
```

will create a link on the text "this is a menu link". When the mouse is moved over this link, the text "click to see menu" will be displayed. If you click on the link, a menu with 3 items will be displayed. If you select "Item 1", the "Command1" is sent to the MUD.

<SEND> to command line

To send text to the client's command line instead of sending it directly to the MUD, use the PROMPT argument in the <SEND> tag.
Example: <SEND "tell Zugg " PROMPT>Zugg</SEND>
will put the text "tell Zugg " (without the quotes) into the command line when the "Zugg" link is clicked.

<EXPIRE [Name]>

The <EXPIRE> tag is used to remove links previously displayed with the <A> or <SEND> tags. For example, when moving to a new room, <SEND> links from the previous room description are no longer valid and need to be removed. To accomplish this, you create a Name for the tags that you want removed. Then you specify this name in the <EXPIRE> tag.

Example:

```
<SEND EXPIRE="Exits">E</SEND>
<EXPIRE Exits>
```

will remove all links that used the EXPIRE="Exits" option in either the <SEND> or <A> tags.

Using <EXPIRE> without any Name argument will expire any <SEND> or <A> tags no matter what Expire Name was given, as long as some name was given. <SEND> and <A> tags that never specified an Expire Name never expire.

Version Control

<VERSION>

The <VERSION> tag can be sent by the MUD to request the version of MXP and the MUD client being used. The client sends the version information back to the MUD in the format of a SECURE <VERSION> MXP tag. The following syntax is sent by the client back to the MUD server:

```
<VERSION MXP=mxpversion STYLE=styleversion CLIENT=clientname VERSION=clientversion REGISTERED=yes/no>
```

The "mxpversion" is the version of the MXP spec implemented by the client. For example, MXP=0.4

The "styleversion" is the current version of the optional style sheet. A MUD sets a style-sheet version number by sending the <VERSION styleversion> tag to the client. The client caches this version information and returns it when requested by a plain <VERSION> request.

The "clientname" is the name of the MUD client. For example, CLIENT=zmud

The "clientversion" is the version of the MUD client. For example, VERSION=6.07

The REGISTERED argument is optional and can be used to detect if the player is using a registered version of the client.

Other arguments can be added to this TAG as needed without breaking older versions.

NOTE: The <VERSION> tag is sent back to the MUD as a SECURE-tagged line. This means that the MXP command is prefixed by <ESC>[1z and followed by a newline. This information makes it easy for the MUD to determine that this information was sent by the client and not by the player.

<SUPPORT>

The <SUPPORT> tag can be used to determine exactly which tags are supported by the client. The client returns this information to the MUD in the form of a SECURE <SUPPORTS> tag (notice the spelling difference).

To determine all of the tags supported by the client, just send <SUPPORT> with no arguments. An example of the text returned to the MUD is:

```
<SUPPORTS +B +I +COLOR>
```

To show that the only MXP tags supported by the client are the <B>, <I>, and <COLOR> tags.

To determine if a specific tag is supported, use it as the argument to the <SUPPORT> tag. For example, to determine if the client supports the <IMAGE> tag, send <SUPPORT image> and the client will return <SUPPORTS +image> if the image tag is supported, or will return <SUPPORTS -image> if the tag is not supported.

You can query an entire list of tags at once by adding them to the tag. For example, to query both the IMAGE tag and the FRAME tag, you can send <SUPPORT image frame> and the client would return <SUPPORT +image -frame> if it supported image but did not support frame.

You can also query the arguments of a tag. For example, to determine the arguments for the <color> tag, use the syntax <SUPPORT "color.*"> The * asks for all arguments, and the client should send back: <SUPPORTS +color.fore +color.back> to indicate that the two arguments supported for the <color> tag are FORE and BACK. To query a specific argument, replace * with the argument name. For example: <SUPPORT color.fore> would return <SUPPORTS +color.fore>.

You can mix and match any of these in the list of arguments. For example:

```
<SUPPORT "color.*" send.expire image>
```

might return: <SUPPORTS +color.fore +color.back -send.expire -image>

It is very important to implement the <SUPPORT> tag in your MUD server. This is the only way to determine what tags are really supported by the client. The MXP version number isn't enough information to determine this, and using <SUPPORT> is easier and better defined than trying to decide based upon what MUD client is being used.

Optional Tags

The tags given in this section may or may not be implemented by MXP clients. Use the <SUPPORT> tag to determine if the client implements the tag. In general, this section provides ideas for client developers interested in extending the MXP concepts, without describing tags that are required to be part of the implementation.

Other HTML tags

The following additional HTML tags might be supported within MXP:

<H1> Level 1 heading
<H2> Level 2 heading
<H3> Level 3 heading
<H4> Level 4 heading
<H5> Level 5 heading
<H6> Level 6 heading
<HR> Horizontal rule
<SMALL> Small text
<TT> Non-proportional font

MSP Compatibility

For MSP Compatibility, the <SOUND> and <MUSIC> tags are available in MXP. They have the same keyword syntax and the equivalent !!SOUND and !!MUSIC MSP tags.

Using Entities

The MUD can instruct the client to automatically create a graphical Gauge or Status Bar text from MXP entities.

<GAUGE EntityName [Max=EntityName] [Caption=text] [Color=color]>

Creates a graphical gauge. The value of the first Entity is used as the gauge data. The Max= argument specifies the name of the entity to use for the maximum value of the gauge. The Caption= text is optionally displayed on the gauge by the client. The Color gives the color of the gauge bar.

<STAT EntityName [Max=EntityName] [Caption=text]>

Creates a new Status Bar text on the client. Text is appended to current status bar. The value of the entity given by the first EntityName is displayed. If the Max= argument is given and the entity named by the Max= argument is Published, then the client displays the text as nnn/mmm where nnn is the first entity value, and mmm is the Max entity value.

Frames

HTML allows you to define regions of your browser window called frames and can direct a separate HTML document to each frame. With MXP, everything needs to be specified from the single MUD text stream, so the normal syntax for frames in HTML does not work for MXP.

Defining a frame in MXP is similar to HTML, but different enough that you should note the syntax. First, the <FRAME> and <FRAMESET> tags are effectively combined. Then, instead of a SRC URL link for the frame, the frame is simply given a name.

In version 1.0 of the MXP spec, the <FRAME> tag was changed to be easier to implement and closer to the <xch_pane> tag used in Pueblo. The concept of a "Parent" for the frame was removed. Frames now default to new windows. The "internal" option can be used to specify a frame within the current MUD window if supported by the client.

The attributes for the <FRAME> tag are:

NAME
: The name of the frame to be used to send text to the frame later in the text stream. Several special names are recognized: _top specifies the main MUD window, _previous specifies the window that was active before this frame.

ACTION (OPEN|CLOSE|REDIRECT)
: The default action is OPEN. An action of OPEN creates the specified window. An action of CLOSE closes a window with the given name that was previously created. An action of REDIRECT will redirect all subsequent MUD output to the specified window. If an action of REDIRECT is used and the frame doesn't exist, it is created first as if an action of OPEN was specified, then output is redirected after the frame is created.

TITLE
: Specifies the full caption of the frame. By default, the Name is used for the Title.

PARENT
: **OBSOLETE**: This attribute was removed in v1.0 of the MXP spec.

INTERNAL
: Specifies that the frame is internal to the current MUD window. The ALIGN attribute specifies how the frame is docked with the MUD window. If this attribute is omitted, a floating frame is created (this is the default).

ALIGN (LEFT|RIGHT|BOTTOM|TOP)
: The alignment of the frame within the parent MUD window. Default is Top. This attribute is ignored unless the INTERNAL attribute is specified.

LEFT
: The coordinate of the left side of the frame. It can be a percentage (ends in %), a number of character widths (ends in "c") or a number of pixels. For example, LEFT="50%" starts the frame halfway across the screen, LEFT="5c" starts the frame 5 character spacings (using the width of the character X if it is a proportional font) from the left side of the screen. LEFT=100 starts the frame 100 pixels from the left edge of the screen. If a negative number is used, it means the value is relative to the right side of the screen instead of the left. Default is 0. Ignored for Internal windows.

TOP
: The coordinate of the top of the frame. Same syntax as LEFT, but the character spacing using the height of the capital X character instead of the width. If a negative value is used, it means the frame is relative to the bottom of the screen instead of the top. Default is 0. Ignored for Internal windows.

WIDTH
: The width of the frame. Same syntax as LEFT. Percentage refers to the percentage of screen width. Ignored for Internal windows.

HEIGHT
: The height of the frame. Same syntax as TOP. Percentage refers to the amount of percentage of the screen height. Ignored for Internal windows.

SCROLLING (yes|no)
: Determines whether the frame is allowed to scroll. The default is NO.

FLOATING
: Forces the frame to "stay on top" of the main MUD window. Ignored for Internal windows.

Some examples:

```
<FRAME Name="Status" Height="2c">
```

Creates a two-line high status window.

```
<FRAME Name="Map" Left="-20c" Top="0" Width="20c" Height="20c">
```

Creates a frame in the upper-right corner of the screen 20 characters wide by 20 characters high.

```
<FRAME Name="Tells" INTERNAL Align="top">
```

Creates a new window called Tells docked to the top of the current MUD window. You cannot control the size of docked windows, as it is client implementation specific.

```
<FRAME Tells REDIRECT>
```

Start redirecting all output to the Tells window

```
<FRAME _previous REDIRECT>
```

Redirect all MUD output to the previous window.

You can also send text to a frame using the <DESTINATION> tag (or <DEST> for short). For example, to send a status message to the Status frame, you would use:

```
<DEST Status>This text is sent to the status window</DEST>
```

If is VERY important to remember the ending </DEST> tag. If the </DEST> tag is not given, the text from the MUD can be lost and never displayed anywhere.

Cursor Control

The <DEST> tag can also be used to position text at a certain position in a frame. So, instead of using VT100 or ANSI escape codes to move the cursor, you can just use the X and Y attributes of the <DEST> command. If no frame name is given, the text is sent to the position on the main MUD window. Note that when text in a frame or window scrolls, the text is no longer at the same X or Y position. So, for status windows, ensure that you set the frame to be unscrollable.

```
<DEST Status X=10 Y=2>100</DEST>
```

Would display the text "100" at the 10 column on the 2nd line of the Status frame.

To erase text already in the frame, you can output spaces, or you can use two of the erase attributes for the <DEST> tag. The EOL keyword causes the rest of the line to be erased after displaying the text. The EOF keyword causes the rest of the frame to be erased after displaying the text. You can erase an entire line using the Y= parameter with no text. You can erase the entire frame by just omitting any text and other keywords. So,

```
<DEST Status Y=2 EOL></DEST>
```

would erase the 2nd line in the status frame, while

```
<DEST Status EOF></DEST>
```

would erase the entire status frame.

Crosslinking multiple MUD servers

Implementation Note (MXP v0.3): The function of the <RELOCATE> command has been changed, and the previous <LOGIN> tag has been replaced by <USER> and <PASSWORD>.

MXP also supports commands that allow multi-server MUD worlds to be built. The <RELOCATE> tag closes the current MUD connection and causes a new connect to open on a new server. The attributes for this tag are the hostname and port of the new connection:

```
<RELOCATE new.server.com 1000>
```

The optional keyword QUIET can be used to suppress further output from the MUD. When the closing </RELOCATE> tag is used, MUD output is resumed. When using the QUIET tag you'll normally put the </RELOCATE> tag after your login sequence to prevent the user from seeing the login text.

To automatically log the user into the other server, use the <USER> tag to cause the client to send the username, and the <PASSWORD> tag to cause the client to send the password.

You can also use <USER> and <PASSWORD> without using the <RELOCATE> command. Putting these two tags in your login sequence will cause MXP-enabled clients to automatically send the user's name and password, instead of relying on client-side scripting for autologin.

Images

MXP supports an <IMAGE> tag to display inline graphics images. GIF or BMP images are supported. Additional image types or encrypted images can be supported with MUD-specific Filters as described in the next section.

This is similar to the <IMG> HTML tag but doesn't implement some of the HTML options. The <IMAGE> tag is very similar to the <SOUND> tag (or !!SOUND command) in MSP. The attributes of the <IMAGE> tag are:

FName
: The name of the graphics file to display.

URL
: The URL of the path for the graphic if it should be downloaded on the fly. This is not recommended for large graphics but is supported to be compatible with the MSP URL parameter for sounds. The classname is appended to the URL, along with the name of the graphics file itself.

T (type)
: The class for the image. Same as classes used in MSP.

H (height)
: The height of the image (in pixels, character heights, or percentage). If omitted, the height is computed from the actual image. If the specified height is different from the image, the image is stretched.

W (width)
: The width of the image. Same syntax as height.

HSPACE
: Additional space to add before and after the image

VSPACE
: Additional space to add above and below the image.

ALIGN (left|right|top|middle|bottom)
: Controls the alignment of the image on the line. For example, if ALIGN=Bottom is used, the rest of the text on the line will align with the bottom of the image.

ISMAP
: Indicates that the image is an image-map. When an image-map is included within a <SEND> tag, the command sent to the MUD is appended with "?X,Y" where X,Y is the position clicked on the image.

Note that when using images, the act as large individual characters as far as the client is concern. Text cannot wrap on multiple lines next to an image...an image always takes up the entire height of the line. Text wrapping to the next line will wrap at the normal left margin of the screen until the graphics image. So, if inline graphics are used, you should make them similar in size to the surrounding text line. Also remember that the user might be using a different screen font.

The <IMAGE> command is intended to be used with graphic images already loaded to the client's disk. Using the URL attribute to download graphics on the fly can cause a severe burden to the MUD server. However, even if the URL attribute is used, the image file is not downloaded if it already exists on the disk. So, since the image is only downloaded once,

the MUD server might be able to handle it. Also, as with MSP, the downloading will never overwrite an existing file with the same name as a security precaution, or allow any absolute path information. The file will be stored in a location determined by the client. To update an image to a new version, you need to specify a new filename.

To implement a clickable image map (server-side map), include an image with the ISMAP attribute within a <SEND> tag. For example:

```
<SEND showmap><IMAGE map.jpg ISMAP></SEND>
```

will create a clickable image map of the map.jpg graphic. When the user clicks on the image, the command given in <SEND> will be sent to the MUD with the ?X,Y appended to it. So, for example, if the user clicked on the X=10, Y=20 point on the image, the command:

```
showmap?10,20
```

will be sent to the MUD.

File Filters

It is likely that the MUD does not want all of the images stored on the clients disk to be viewable until the MUD server requests them. MXP allows the MUD to define it's own graphics format and to provide a client plugin module that converts the MUD-specific format to a standard GIF or BMP format. To define a filter, use the <FILTER> tag, with the SRC attribute to specify the file extension of the MUD-specific format, and the DEST attribute to specify the output file extension (default is BMP). The NAME attribute gives the name of the plugin to be called, and the optional PROC attribute is a numeric parameter that the plugin can use to support multiple conversions as needed. So, an example would be:

```
<FILTER SRC='gff' DEST='gif' NAME='MyPlugin'>
```

This would call the Filter function in the MYPLUGIN plugin module whenever the MUD tries to display a .gff file. The .gff file is read into memory and passed to the Filter function in the plugin (with PROC=0 in the above example) and expect a .gif format output to be returned by the function. Simply provide your plugin for users and you can then store proprietary files on the client that are decoded and displayed as needed. Note that this same Filter function can also be used to process sound files used by MSP.

Outdated Tags

Server-side scripting

The <SCRIPT> tag is no longer supported in version 0.4 MXP. There were too many client-specific security issues involved in this tag.

Implementation Details

To implement MXP on a MUD server requires some attention. Most important, you need some way to activate the MXP text output on the MUD. You don't want to display the MXP tags to a user that doesn't have an MXP MUD client. One way to activate MXP on your MUD is to simply prompt the user and ask if they want to enable MXP, or provide an MXP command on the MUD that toggles MXP text on and off.

If you want to get more sophisticated, you can use the Telnet Option negotiation to determine if the client supports MXP. Simple query Telnet option 91 and an MXP client will respond that it will handle MXP. You can also use the Telnet option to determine if the client supports the MUD Sound Protocol (MXP) or the MUD Compression Protocol (MCP). The following Telnet Option numbers are used:

86 MUD Compression Protocol (see MCP spec for details on the difference between option 85 and option 86)
90 MUD Sound Protocol (MSP)
91 MUD eXtension Protocol (MXP)

Although MXP does not support a separate style sheet, you would be best off in your implementation if you stored all of your !ELEMENT and !ENTITY definitions in a separate file and sent them to the client as part of your welcome message. This would give you one place to change your tags on the MUD. The MUD output itself would then use these elements. You should try to avoid using MXP commands within the specific room output of a MUD and instead encapsulate them into MUD-defined ELEMENTS so that you can easily change how your MUD looks and operates.

A Detailed Example

Want a more complete example of all of this? I thought so. Here is a basic MXP package the you can use on your MUD to handle the most common MUD functions. Go through this example line by line with the documentation above and you should be able to figure it out. Once you get the hang of it, it is pretty easy.

```
<!-- Elements to support the Automapper -->
<!ELEMENT RName '<FONT COLOR=Red><B>' FLAG="RoomName">
<!ELEMENT RDesc FLAG='RoomDesc'>
<!ELEMENT RExits '<FONT COLOR=Blue>' FLAG='RoomExit'>
<!-- The next element is used to define a room exit link that sends
the exit direction to the MUD if the user clicks on it -->
<!ELEMENT Ex '<SEND>'>
```

```
<!ELEMENT Chat '<FONT COLOR=Gray>' OPEN>
<!ELEMENT Gossip '<FONT COLOR=Cyan>' OPEN>
<!-- in addition to standard HTML Color specifications, you can use
color attribute names such as blink -->
<!ELEMENT ImmChan '<FONT COLOR=Red,Blink>'>
<!ELEMENT Auction '<FONT COLOR=Purple>' OPEN>
<!ELEMENT Group '<FONT COLOR=Blue>' OPEN>
<!-- the next elements deal with the MUD prompt -->
<!ELEMENT Prompt FLAG="Prompt">
<!ELEMENT Hp FLAG="Set hp">
<!ELEMENT MaxHp FLAG="Set maxhp">
<!ELEMENT Mana FLAG="Set mana">
<!ELEMENT MaxMana FLAG="Set maxmana">
<!-- now the MUD text -->
<RName>The Main Temple</RName>
<RDesc>This is the main hall of the MUD where everyone starts.
Marble arches lead south into the town, and there is a <i>lovely</i>
<send "drink &text;">fountain</send> in the center of the temple,</RDesc>
<RExits>Exits: <Ex>N</Ex>, <Ex>S</Ex>, <Ex>E</Ex>, <Ex>W</Ex></RExits>

<Prompt>[<Hp>100</Hp>/<MaxHp>120</MaxHp>hp <Mana>50</Mana>/<MaxMana>55</MaxMana>mana]</Prompt>
```

This would display the following:

The Main Temple
This is the main hall of the MUD where everyone starts.
Marble arches lead south into the town, and there is a *lovely*
fountain in the center of the temple.
Exits: N, S, E, W

[100/120hp 50/55mana]

and set the variables @hp, @maxhp, @mana, @maxmana appropriately. Clicking on the exit links would send the proper direction to the MUD to move you in that direction. It would send the "drink fountain" command if the user clicked on the underlined word "fountain".

Conclusions

Proper use of MXP can radically improve the normal text-only MUD experience. In addition to embedded graphics, you can now better control the font, color, and size of text. You can directly control the automapper, allowing it to work perfectly on your MUD without fancy parsing. You can define links for users that like to point and click with their mouse. You can add pull-down menus to text displayed by the MUD to give the user a list of possible commands. The use of MXP will make your MUD stand out among the hundreds of other MUDs on the Internet and help breath new life into MUDs and help them compete against the more graphical RPGs appearing on the Internet.