1 Overview

Despite the availability of Unicode, text encoders still sometimes find that the published repertoire of available characters and glyphs is inadequate to their needs. This is particularly the case when dealing with ancient languages, for example. These Guidelines provide a mechanism to satisfy that need, while retaining compatibility with standards.

If encoders encounter some graphical unit in a document they want to render electronically, the first question that needs to be asked is: ‘Is this a character?’ To determine whether a particular graphical unit *is* a character or not, see Terminology and key concepts.

If the unit is indeed determined to be a character, the next question is, ‘Has this character been encoded already?’ In order to determine if a character has been encoded, users should follow the following steps:

1. Check the Unicode website, first reading the webpage Where is my Character?", then the code charts). Alternatively, users can check the latest published version of The Unicode Standard, though the website is often more up to date than the printed version, and should be checked for preference.

   The pictures (‘glyphs’) in the Unicode code charts are only meant to be representative, not definitive. If a specific form of an already encoded character is required for a project, refer to the guidelines contained below under Annotating Characters.

2. Check the Proposed New Characters webpage (`http://unicode.org/alloc/Pipeline.html`) to see whether the character is in line for approval.

3. Ask on the Unicode email list to determine if a proposal is pending, or whether this is indeed a new character (or if this not a character at all, in which case it would not be eligible for addition to the Unicode standard).

Since there are now close to 100,000 characters in Unicode, chances are good that what you need is already there, but it might not be easy to find, since it might have a different name in Unicode. Try a bit harder and use other sites, for example `http://www.eki.ee/letter`, which allows also searches based on scripts and languages.

An encoded character may be precomposed or it may be formed from base characters and combining diacritical marks. Either will suffice for a character to be "found" as an encoded character.

If this first question has been considered and no suitable form has been found in such a repertoire, the next question will be: ‘Does the graphical unit in question represent a variant form of a known character, or does it represent a completely unencoded character?’ If the character is determined to be missing from Unicode, it would be helpful to submit the new character for inclusion (see `http://unicode.org/pending/proposals.html`).

These guidelines will help you proceed once you have identified a given graphical unit as either a variant or an unencoded character. Determining this will require knowledge of the contents of the document that you have. The first case will be called *annotation* of a character, while the second case will be called *adding* of a new character. How to handle graphical units that represent variants will be discussed below (3. Annotating characters) while the problem of representing new characters will be dealt with in section 4. Adding new characters.

While there is some overlap between these requirements, distinct specialized markup constructs have been created for each of these cases as explained in section 2. Markup constructs for representing non-standard characters below. The following section will then proceed to discuss how to apply them to the problems at hand, discussing annotation of existing characters in section 3. Annotating characters and finally creation of new ones in 4. Adding new characters.

2 Markup constructs for representing non-standard characters

The module provides a mechanism to declare characters additional to those available from the document character set. XML allows for a document (or document component) to declare that it uses only a named subset of Unicode characters: this is specified by the encoding parameter on the XML declaration. In such a case, it might be necessary to declare as additional characters some characters already defined by Unicode, but not available in the named subset. Generally speaking, however, the document character set will be Unicode, and this mechanism will be needed only for characters not defined by Unicode.

The mechanism provided here consists functionally of two parts:

- an element `<c>`, which serves as a proxy for new characters or glyphs

- an element `<charDesc>`, which serves as a container for information about this character. Such information is needed to identify and process this character properly.

When the module is included in a TEI schema, the `<charDesc>` element is added to the **`tei.encoding`** class, and the `<c>` element is added to the phrase class. These elements and their components are documented in the rest of this section.

The Unicode standard defines properties for all the characters it defines in the the Unicode Character Database, knowledge of which is usually built into text processing systems. Since the characters that are instantiated with the `<c>` element either do not exist in Unicode at all, or are not available in the subset of Unicode in use, their properties are not readily available. The `<charDesc>` element makes it possible to store such properties for use by applications in a standard way.

The list of attributes (properties) for characters is modelled on those in the Unicode Character Database, which distinguishes normative and informative character properties. Additional, non-Unicode, properties may also be supplied. Since the list of properties will vary with different versions of The Unicode Standard, there may not be an exact correspondence between them and the list of properties defined in these Guidelines.

Usage examples for these elements are given below at 3. Annotating characters and 4. Adding new characters. The module itself is formally defined in section 6. Additional module defined by this section below. It defines the following additional elements:

- <**charDesc**> provides descriptive information about characters or glyphs additional to that implied by the current document character set.

No attributes other than those globally available (see definition for tei.global.attributes)

- <**c**> represents a character.

ref	identifies the character or glyph description intended.

The following elements may appear within a `<charDesc>` element:

-

- <**char**> provides descriptive information about a character which is not otherwise available in the document character set.

ucs	specifies the codepoint used for this character in the current document.

- <**glyph**> provides descriptive information about a character glyph which is not otherwise available in the document character set.

ucs	specifies the codepoint used for this glyph in the current document.

The `<char>` and `<glyph>` elements have similar contents and are used in similar ways, but their functions are different. The `<char>` element is provided to define a character which is not available in the current document character set, for whatever reason, as stated above. The `<glyph>` element is used to annotate an existing character, usually by providing a specific glyph that shows how a character appeared in the original document. Unicode codepoints refer to a very general abstract character, which can be rendered by a very large number of possible representations. The `<glyph>` element is provided for cases where the encoder wants to specify a specific glyph (or family of glyphs) out of all possible glyphs.

As noted above, the `ucs` attribute specifies the codepoint used for the character or glyph under discussion. Its value must match the pattern **`datatype.ucs`**, which is also defined by this module. Values matching this pattern take the form `U+xxxx`, where `xxxx` is the hexadecimal representation of any valid Unicode code point, including values taken from the 'Private Use Area' (see further 5. How to use codepoints from the Private Use Area).

The Unicode Standard recommends naming conventions which must be followed strictly where the intention is to annotate an existing Unicode character, and which should also be used as a model when creating new names for characters or glyphs. For convenience of processing, the following distinct elements are proposed for naming characters and glyphs:

- <**charName**> contains the name of a character, expressed following Unicode conventions.

No attributes other than those globally available (see definition for tei.global.attributes)

- **<glyphName>** contains the name of a glyph, expressed following Unicode conventions.

No attributes other than those globally available (see definition for tei.global.attributes)

Within both `<char>` and `<glyph>`, the following elements are available:

-
- **<charProp>** provides a name and value for some property of the parent character or glyph.

No attributes other than those globally available (see definition for tei.global.attributes)

-
- **<mapping>** contains one or more characters which are related to the parent character or glyph in some respect, as specified by the type attribute.

No attributes other than those globally available (see definition for tei.global.attributes)

-
-

Four of these elements (`<gloss>`, `<desc>`, `<figure>` and `<remarks>`) are defined by other TEI modules, and their usage here is no different from their usage elsewhere. The `<figure>` element, however, is used here only to link to an image of the character or glyph under discussion, or to contain a representation of it in SVG. Several `<figure>` elements may be given, for example to provide images with different resolution, or in different formats. The `type` attribute may be used to specify the type of image.

The `<mapping>` element is similar to the standard TEI `<equiv>` element. While the latter is used to express correspondence relationships between TEI concepts or elements and those in other markup vocabularies, the former is used to express any kind of relationship between the character or glyph under discussion and characters or glyphs defined elsewhere. It may contain any Unicode character, or a `<c>` element linked to some other `<char>` or `<glyph>` element, if, for example, the intention is to express an association between two non-standard characters. The type of association is indicated by the `type` attribute, which may take such values as `exact` for exact equivalences, `uppercase` for uppercase equivalences, `lowercase` for lowercase equivalences, `standardized` for standardized forms, and `simplified` for simplified characters, etc., as in the following example:

```
<charDesc>
<char id="aenl" ucs="U+E300">
<charName>LATIN LETTER ENLARGED SMALL A</charName>
<charProp>
<localName>entity</localName>
<value>aenl</value>
<mapping type="standardized">a</mapping>
</char>
</charDesc>
```

A more precise documentation of the properties of any character or glyph may be supplied using the generic `<charProp>` element described in the next section. Despite its name, this element may be used for either characters or glyphs.

2.1 Character Properties

The Unicode Standard documents ‘ideal’ characters, defined by reference to a number of properties (or attribute-value pairs) which they are said to possess. For example, a lower case letter is said to have the value `Ll` for the property `general-category`. The Standard distinguishes between normative properties (i.e. properties which form part of the definition of a given character), and informative or additional properties which are purely descriptive. It also allows for the addition of new properties, and (in some circumstances) alteration of the values currently assigned to certain propertiesSee further Character Property Model

The `<charProp>` element allows encoder to supply information about a character or glyph. Where the information concerned relates to a property which has already been identified in The Unicode Standard, encoders are urged to use the appropriate Unicode name.

The following elements are used to record character properties:

- **<unicodeName>** contains the name of a registered Unicode normative or informative property.

No attributes other than those globally available (see definition for tei.global.attributes)

- **<localName>** contains a locally defined name for some property.

No attributes other than those globally available (see definition for tei.global.attributes)

- **<value>** contains a single value for some property, attribute, or other analysis.

No attributes other than those globally available (see definition for tei.global.attributes)

For each property, the encoder must supply either a `<unicodeName>` or a `<localName>`, followed by a `<value>`.

We list here some of the more significant normative character properties which may be provided. More information about the normative character properties in Unicode can be found at The Unicode Standard, Version 3.0, Addison and Wesley , p. 73, Table 4-1).

general-category The general category (described in The Unicode Standard 4.5) is an assignment to some major classes and subclasses of characters. Some typical values for this property are listed here:

Lo Letter, other

Ll Letter, lowercase

canonical-combining-class This property exists for characters that are not used independently, but in combination with other characters. It records a class for these characters, which is used to determine how character interact typographically. For more information, see The Unicode Standard 4.2

directional-category All Unicode characters possess a directional type, which governs the application of the algorithm for bi-directional behaviour.

character-decomposition-mapping This is used to determine the relationship to other character(s). The Unicode Standard contains a list of tags used for this purpose.

numeric-value The numeric value (in decimal notation) of a character that expresses any kind of numeric value.

mirrored The mirrored character property is used to properly render characters such as U+0028, `OPENING PARENTHESIS` independent of the text direction.

The Unicode Standard also defines a set of informative (but non-normative) properties for Unicode characters. If encoders want to provide such properties, they may be included using the suggested Unicode name.

Any Unicode-defined property must be named using the `<unicodeName>` element. However, encoders may also supply any locally-defined properties, which must be named using the `<localName>` element to distinguish them. These properties do not parallel properties given in the The Unicode Standard.

3 Annotating characters

Annotation of characters is necessary when a distinction based on only some aspects of a character (such as its graphical appearance) is desired. In a manuscript, for example, where distinctly different forms of the letter "r" can be recognized, it might be useful to distinguish them for analytic purposes, quite distinct from the need to provide a accurate representation of the page. A digital facsimile, particularly one linked to a transcribed and encoded version of the text, will always provide a superior visual representation, but cannot be used to support arguments based on the distribution of such different forms. Character annotation as described here provides a solution to this problem. It should be kept in mind that any kind of text encoding is an abstraction and interpretation of the text at hand and will not be useful in reproducing an exact facsimile of a manuscript.,

We begin by defining two distinct `<glyph>` elements, one for each of the forms of the letter we wish to distinguish:

```
<charDesc>
<glyph id="r1">
<glyphName>LATIN SMALL LETTER R WITH ONE FUNNY STROKES</glyphName>
<charProp>
<localName>entity</localName>
<value>r1</value>
</charProp>
<figure url="r1img.png"/>
</glyph>
<glyph id="r2">
<glyphName>LATIN SMALL LETTER R WITH TWO FUNNY STROKES</glyphName>
<charProp>
<localName>entity</localName>
<value>r2</value>
</charProp>
<figure url="r2img.png"/>
</glyph>
</charDesc>
```

With these definitions in place, occurrences of these two special "r"s in the text can be annotated using the element `<c>`:

```
<p>Wo<c ref="#r1">r</c>ds in this
manusc<c ref="#r2">r</c>ipt are sometimes
written in a funny way.</p>
```

As can be seen in this example, the `<glyph>` element pointed to from the `<c>` element will be interpreted as an annotation on the content of the element `<c>`. This mechanism can also be used to indicate ligatures, as in the following example:

```
<p> ...  <c ref="#Filig">Fi</c>lthy riches...</p>
<!- in the chardesc ->
<glyph id="Filig">
<glyphName>LATIN UPPER F AND LATIN LOWER L LIGATURE</glyphName>
<figure url="Filig.png"/>
</glyph>
```

The same technique might be used to represent particular abbreviation marks, within the scope of an `<abbr>` element, as in the following example:

```
<!- to be supplied ->
```

With this markup in place, it will be possible to write programs to analyze the distribution of the different letters "r" as well as produce more ‘faithful’ renderings of the original. It will also be possible to produce normalized versions by simple ignoring the annotation pointed to by the element `<c>`.
For brevity of encoding, it may be preferred to pre-define internal entities such as the following:

```
<!ENTITY r1 ’<c ref="#r1">r</c>’ >
<!ENTITY r2 ’<c ref="#r2">r</c>’ >
```

which would enable the same material to be encoded as follows:

```
<p>Wo&r1;ds in this manusc&r2;ipt are
sometimes written in a funny way.</p>
```

Since this mechanism employs markup objects to provide a link between a character in the document and some annotation on that character, it cannot be used in places where such markup constructs are not allowed, e.g. in attribute values.

4 Adding new characters

The creation of additional characters for use in text encoding is similar to the annotation of an existing character. The same element `<c>` is used to provide a link from the character instance in the text to the character definition in `<charDesc>` element. The main difference is that the `<c>` element now points to a `<char>` element. The element `<c>` could however also hold a codepoint from the Private Use Area (PUA) of The Unicode Standard, which is an area set aside for the very purpose of privately adding new characters to a document. Recommendations on how to assign such PUA characters are given in the following section.

Under certain circumstances, Han characters can be written within a circle. While this could be considered simply a facet of the rendering, it can also be considered a new, derived character, which will be in many ways similar to the original, non-circled character, but has a distinct rendering. The following example will provide the necessary markup to encode such an encircled character.

```
<char id="U4EBA-circled" ucs="U+E000">
<charName>CIRCLED IDEOGRAPH</charName>
<charProp>
<unicodeName>character-decomposition-mapping</unicodeName>
<value>circle</value>
</charProp>
<charProp>
<localName>daikanwa</localName>
<value>36</value>
</charProp>
<mapping type="standard">
<c ref="U4EBA">&#x4eba;</c>
</mapping>
</char>
```

5 How to use codepoints from the Private Use Area

The developers of the Universal Character Set have set aside an area of the codespace for the private use of software vendors, user groups or individuals. As of this writing (Unicode 4.0), there are around 137,000 codepoints available in this area, which should be enough for most needs. No codepoint assignments will be made to this area by standard bodies and only some very basic default properties have been assigned (which will be overwritten where necessary by the mechanism outlined in this chapter). Therefore, in contrast to all other codepoints of the UCS, PUA codepoints should not be used directly in documents intended for blind interchange. Instead of using PUA codepoints directly in the document content, entity references should be used. This will make it easier for receiving parties to find out what PUA characters are used in a document and where possible codepoint clashes with local use on the receiving side occurs.

To give a concrete example, the third variant of the letter r, `&r3;` in the above example might in the local processing environment be more conveniently assigned a codepoint from the PUA, (say, U+E000), perhaps to facilitate the use of a particular font which displays the desired character at this codepoint. Since this assignment would be valid only on the local site, texts containing such codepoints are not suitable for blind interchange. During the process of preparing these texts for interchange, such PUA codepoints should be changed either to an encoding such as `<c ref="#r2">` or to an entity reference such as `&r2;`, either of which will reference an explicitly defined `<char>` element. The PUA character used during the preparation of the document could be recorded as value of the `ucs` attribute, as shown in the example in 4. Adding new characters. Upon receipt on some other site, these could then again be transformed to characters from the PUA. If, as might well be the case, this other site has already made an assignment of some other character to the codepoint U+E000, it would need to represent the variant letter to some different and hitherto unused codepoint (say, U+E080).

This mechanism is rather weak in cases where DOM trees or parsed XML fragments are exchanged, which might be increasingly the case. The best an application can do here is to treat any occurrence of a PUA character only in the context of the local document and use the properties provided through the `<char>` element as a handle to the character in other contexts.

In the fullness of time, a character may become standardized, and thus assigned a specific code point outside the PUA. Documents which have been encoded using the mechanism must at the least change the value of the `ucs` attribute on the relevant `<char>` element to indicate the new correct codepoint. They may also be modified to replace all occurrences of `<c>` elements which reference this `<char>` element by occurrences of the specific coded character.

6 Additional module defined by this section

The elements described in this section are declared in a module called **`wsd-ng`** which has the following formal declaration:

<!– : Character and Glyph documentation–>

```
namespace local = ""
datatype.ucs = text
```

```
namespace local = ""
c = element c  c.content
c.content = c.attributes, text
c.attributes =
tei.global.attributes,
attribute ref  text ?,
[ :defaultValue = "c" ] attribute TEIform  text ?
```

```
namespace local = ""
char = element char  char.content
char.content =
char.attributes,
( charName, gloss?, charProp*, desc?, mapping*, figure*, note?  )
char.attributes =
tei.global.attributes,
attribute ucs  datatype.ucs ?,
[ :defaultValue = "char" ] attribute TEIform  text ?
```

```
namespace local = ""
charName = element charName  charName.content
charName.content = charName.attributes, text
charName.attributes =
tei.global.attributes,
[ :defaultValue = "charName" ] attribute TEIform  text ?
```

```
namespace local = ""
charProp = element charProp  charProp.content
charProp.content = charProp.attributes, ( ( unicodeName | localName
), value )
charProp.attributes =
tei.global.attributes,
[ :defaultValue = "charProp" ] attribute TEIform  text ?


namespace local = ""
charDesc = element charDesc  charDesc.content
charDesc.content = charDesc.attributes, ( desc?, ( char | glyph )+
)
charDesc.attributes =
tei.global.attributes,
[ :defaultValue = "charDesc" ] attribute TEIform  text ?


namespace local = ""
glyph = element glyph  glyph.content
glyph.content =
glyph.attributes,
( name, gloss?, charProp*, desc?, mapping*, figure*, note?  )
glyph.attributes =
tei.global.attributes,
attribute ucs  datatype.ucs ?,
[ :defaultValue = "glyph" ] attribute TEIform  text ?


namespace local = ""
glyphName = element glyphName  glyphName.content
glyphName.content = glyphName.attributes, text
glyphName.attributes =
tei.global.attributes,
[ :defaultValue = "glyphName" ] attribute TEIform  text ?


namespace local = ""
localName = element localName  localName.content
localName.content = localName.attributes, text
localName.attributes =
tei.global.attributes,
[ :defaultValue = "localName" ] attribute TEIform  text ?
```

```
namespace local = ""
mapping = element mapping  mapping.content
mapping.content = mapping.attributes, ( ( text | c )* )
mapping.attributes =
tei.global.attributes,
[ :defaultValue = "mapping" ] attribute TEIform  text ?
```

```
namespace local = ""
unicodeName = element unicodeName  unicodeName.content
unicodeName.content = unicodeName.attributes, text
unicodeName.attributes =
tei.global.attributes,
[ :defaultValue = "unicodeName" ] attribute TEIform  text ?
```

```
namespace local = ""
value = element value  value.content
value.content = value.attributes, text
value.attributes =
tei.global.attributes,
[ :defaultValue = "value" ] attribute TEIform  text ?
```

<!– end of –>

7 Reference section

[datatype.ucs]

datatype.ucs	(Unicode Codepoint) Any legal Unicode codepoint, expressed as four hexadecimal digits prefixed by the characters `U+`.
Declaration	`namespace local = ""` `datatype.ucs = text`
Module	[wsd-ng]

[<c>]

<c>	(character) represents a character.
Module	[wsd-ng]
May contain	
May occur within	
Declaration	`namespace local = ""` `element c  text, tei.global.attributes, attribute ref  text ?`

Attributes	(In addition to global attributes) `ref` — identifies the character or glyph description intended. *Datatype:*
Example	`<c ref="#flig">fl</c>`
Note	Should only contain a single character or a character entity that represents a single character.

[<char>]

`<char>`	(character) provides descriptive information about a character which is not otherwise available in the document character set.
Module	[wsd-ng]
May contain	
May occur within	

Declaration	`namespace local = ""`
	`element char`
	`( charName, gloss?, charProp*, desc?, mapping*, figure*, note?  ),`
	`tei.global.attributes,`
	`attribute ucs  datatype.ucs ?`

Attributes	(In addition to global attributes) `ucs` — specifies the codepoint used for this character in the current document. *Datatype:* datatype.ucs
Example	

```
<char id="U4EBA-circled" ucs="U+4EBA">
<charName>CIRCLED IDEOGRAPH 4EBA</charName>
<charProp>
<unicodeName>character-decomposition-mapping</unicodeName>
<value>circle</value>
</charProp>
<charProp>
<localName>daikanwa</localName>
<value>36</value>
</charProp>
<mapping type="standard">
<c ref="U4EBA">[U+4EBA]</c>
</mapping>
</char>
```

[<charName>]

`<charName>`	(character name) contains the name of a character, expressed following Unicode conventions.
Module	[wsd-ng]
May contain	
May occur within	

Declaration	`namespace local = ""`
	`element charName  text, tei.global.attributes`

Attributes	Global attributes only
Example	`<charName>CIRCLED IDEOGRAPH 4EBA</charName>`
Note	The name must follow Unicode conventions for character naming. Projects working in similar fields are recommended to coordinate and publish their list of `<charName>`s to facilitate data exchange.

[<charProp>]

`<charProp>`	(character property) provides a name and value for some property of the parent character or glyph.
Module	[wsd-ng]
May contain	
May occur within	

Declaration	`namespace local = ""`
	`element charProp`
	`( ( unicodeName | localName ), value ),`
	`tei.global.attributes`

Attributes	Global attributes only

Example	`<charProp>`
	`<unicodeName>character-decomposition-mapping</unicodeName>`
	`<value>circle</value>`
	`</charProp>`
	`<charProp>`
	`<localName>daikanwa</localName>`
	`<value>36</value>`
	`</charProp>`

Note	If the property is a Unicode Normative Property, then its `<unicodeName>` must be supplied. Otherwise, its name must be specied by means of a `<localname>`.At a later release, additional constraints will be defined on possible value/name combinations using Schematron rules

[<charDesc>]

`<charDesc>`	(character description) provides descriptive information about characters or glyphs additional to that implied by the current document character set.
Module	[wsd-ng]
May contain	
May occur within	

Declaration	`namespace local = ""`
	`element charDesc  ( desc?, ( char | glyph )+ ), tei.global.attributes`

Attributes	Global attributes only
Example	

[<glyph>]

`<glyph>`	(character glyph) provides descriptive information about a character glyph which is not otherwise available in the document character set.
Module	[wsd-ng]
May contain	
May occur within	

Declaration	`namespace local = ""`
	`element glyph`
	`( name, gloss?, charProp*, desc?, mapping*, figure*, note?  ),`
	`tei.global.attributes,`
	`attribute ucs  datatype.ucs ?`

Attributes	(In addition to global attributes) `ucs` — specifies the codepoint used for this glyph in the current document. *Datatype:* datatype.ucs

Example	`<glyph id="r1">`
	`<name>LATIN SMALL LETTER R WITH A FUNNY STROKE</name>`
	`<charProp>`
	`<localName>entity</localName>`
	`<value>r1</value>`
	`</charProp>`
	`<figure url="glyph-r1.png"></figure>`
	`</glyph>`

[<glyphName>]

`<glyphName>`	(character glyph name) contains the name of a glyph, expressed following Unicode conventions.

Module	[wsd-ng]
May contain	
May occur within	

Declaration	`namespace local = ""`
	`element glyphName  text, tei.global.attributes`

Attributes	Global attributes only
Example	`<glyphName>CIRCLED IDEOGRAPH 4EBA</glyphName>`
Note	For characters of non-ideographic scripts, a name following the conventions for Unicode names should be chosen. For ideographic scripts, an Ideographic Description Sequence (IDS) as described in Chapter 10.1 of The Unicode Standard is recommended where possible. Projects working in similar fields are recommended to coordinate and publish their list of `<glyphName>`s to facilitate data exchange.

[<localName>]

`<localName>`	(Locally-defined Property Name) contains a locally defined name for some property.
Module	[wsd-ng]
May contain	
May occur within	

Declaration	`namespace local = ""`
	`element localName  text, tei.global.attributes`

Attributes	Global attributes only

Example	`<localName>daikanwa</localName>`
	`<localName>entity</localName>`

Note	No definitive list of local names is proposed. However, the name **`entity`** is recommended as a means of naming the property identifying the recommended character entity name for this character or glyph.

[<mapping>]

`<mapping>`	(character mapping) contains one or more characters which are related to the parent character or glyph in some respect, as specified by the `type` attribute.
Module	[wsd-ng]
May contain	
May occur within	

Declaration	`namespace local = ""`
	`element mapping  ( ( text | c )* ),`
	`tei.global.attributes`

Attributes	Global attributes only

Example	`<mapping type="modern">r</mapping>`
	`<mapping type="standard">`
	`<c ref="U4EBA">[U+4EBA]</c>`
	`</mapping>`

Note	Suggested values for the `type` attribute include `exact` for exact equivalences, `uppercase` for uppercase equivalences, `lowercase` for lowercase equivalences, and `simplified` for simplified characters. The `<c>` elements contained by this element can point to either another `<char>` or `<glyph>`element or contain a character that is intended to be the target of this mapping.

[<unicodeName>]

`<unicodeName>`	(Unicode Property Name) contains the name of a registered Unicode normative or informative property.
Module	[wsd-ng]
May contain	
May occur within	

Declaration	`namespace local = ""`
	`element unicodeName  text, tei.global.attributes`

Attributes	Global attributes only

Example	`<unicodeName>character-decomposition-mapping</unicodeName>`
	`<unicodeName>general-category</unicodeName>`

Note	A definitive list of current Unicode property names is provided in The Unicode Standard.

[<value>]

`<value>`	(value) contains a single value for some property, attribute, or other analysis.
Module	[wsd-ng]
May contain	
May occur within	

Declaration	`namespace local = ""`
	`element value  text, tei.global.attributes`

Example	`<value>unknown</value>`